Some Restricted Circuit Classes that Can Be as Inefficient as DNFs

Diplom Thesis

by

Matthias P. Krieger

June 12, 2006

Submitted to the Department of Mathematics

at Johann Wolfgang Goethe-Universität, Frankfurt am Main

Abstract (in German)

Diese Arbeit beschäftigt sich mit Booleschen Schaltkreisen, die aus UND-, ODER- und NICHT-Gattern aufgebaut sind. Wenn nicht anders angegeben, setzen wir dabei voraus, dass jedes Gatter höchstens zwei Eingänge hat. Boolesche Schaltkreise berechnen Boolesche Funktionen. Die *Schaltkreiskomplexität* einer Booleschen Funktion f ist die kleinstmögliche Zahl von Gattern eines Schaltkreises für f.

Die beste bekannte untere Schranke für die Schaltkreiskomplexität einer explizit angegebenen Funktion ist noch linear in der Zahl der Variablen. Es konnten jedoch superpolynomiale untere Schranken für *monotone* Schaltkreise bewiesen werden. Ein Schaltkreis heißt monoton, wenn er keine NICHT-Gatter (Negationen) aufweist. Die Approximationsmethode liefert superpolynomiale untere Schranken für die monotone Schaltkreiskomplexität verschiedener Funktionen. Auch ist bekannt, dass Negationen zur Berechnung so genannter Slice-Funktionen fast keinen Beitrag leisten können. Eine superpolynomiale untere Schranke für die monotone Komplexität einer Slice-Funktion impliziert eine superpolynomiale untere Schranke für ihre nicht-monotone Komplexität. Allerdings reichen die heute bekannten Methoden anscheinend nicht aus, um ausreichende untere Schranken für die Komplexität von Slice-Funktionen zu zeigen. Daher ist es gerechtfertigt, nach neuen Ansätzen für den Beweis unterer Schranken monotoner Schaltkreiskomplexität zu suchen.

In dieser Arbeit unternehmen wir einige Schritte in diese Richtung. Wir untersuchen einige Schaltkreisklassen, die eingeschränkter sind als monotone Schaltkreise. Wir beweisen optimale exponentielle untere Schranken für diese Schaltkreisklassen. Sie sagen aus, dass die Schaltkreise genauso viele ODER-Gatter benötigen wie die disjunktiven Normalformen der untersuchten Funktionen. Das heißt, dass wir kein einziges ODER-Gatter sparen können, wenn wir die jeweiligen Schaltkreisklassen anstelle von disjunktiven Normalformen zulassen. Wir sagen deshalb, diese disjunktiven Normalformen sind *unkomprimierbar*. Wir betrachten diese Unkomprimierbarkeit als eine bemerkenswerte Eigenschaft der untersuchten Funktionen. Wir beweisen auch eine obere Schranke, die zeigt, dass einige dieser disjunktiven Normalformen im Falle allgemeiner monotoner Schaltkreise doch stark komprimierbar sind.

Wir führen Pseudo-Slice-Funktionen ein, die Slice-Funktionen ähnlich sind. Da

keine untere Schranken für Slice-Funktionen bekannt sind, schlagen wir vor, die Komplexität der Pseudo-Slice-Funktionen zu untersuchen. Die Beweismethoden für untere Schranken, die wir in dieser Arbeit vorstellen, sind auch auf Pseudo-Slice-Funktionen anwendbar.

Wir nennen einen Schaltkreis multilinear, wenn die Eingänge zu jedem seiner UND-Gatter aus disjunkten Variablenmengen berechnet werden. Um eine formalere Definition anzugeben, bezeichnen wir mit $var(g)$ die Menge der Variablen, die im bei g wurzelnden Teilschaltkreis vorkommen. Ein Schaltkreis ist multilinear, wenn $var(g_1) \cap var(g_2) = \emptyset$ für jedes seiner UND-Gatter mit Eingängen g_1 und g_2. Multilineare Schaltkreise wurden in [40, 27] untersucht ([27] verwendet eine etwas weniger einschränkende Definition der Multilinearität). Multilineare Schaltkreise sind eine Verallgemeinerung nichtdeterministischer read-once Branching-Programme, die viel Aufmerksamkeit auf sich gezogen haben.

Wir bezeichnen mit $PI(f)$ die Menge der Primimplikanten einer Booleschen Funktion. Es ist klar, dass jede Funktion f von einem multilinearen Schaltkreis mit $|PI(f)| - 1$ ODER-Gattern berechnet werden kann: man nehme einfach die disjunktive Normalform. Viele bekannte Funktionen haben multilineare Schaltkreise, die viel kleiner sind als ihre disjunktiven Normalformen. Die Threshold-Funktion T_k^n ist ein Beispiel. Die Threshold-Funktion T_k^n hat $\binom{n}{k}$ Primimplikanten, kann aber von einem multilinearen Schaltkreis der Größe $O(nk)$ berechnet werden. Die Lücke zwischen der Größe des kleinsten multilinearen Schaltkreises für eine gewisse Funktion und die Größe der disjunktiven Normalform dieser Funktion kann also exponentiell sein. Es ist auch bekannt, dass die Lücke zwischen der multilinearen Komplexität und der monotonen Komplexität exponentiell ist [40].

Wir finden eine Klasse von Funktionen, deren multilineare Schaltkreise genauso viele ODER-Gatter wie ihre disjunktiven Normalformen benötigen, nämlich die so genannten vereinigungsfreien Funktionen. Wir nennen eine monotone Funktion *vereinigungsfrei*, wenn die Vereinigung von zwei Primimplikanten nie einen weiteren Primimplikanten enthält. Wir beweisen den folgenden Satz.

Satz 5.4. *Sei f eine monotone vereinigungsfreie Funktion. Dann hat jeder multilineare Schaltkreis für f mindestens $|PI(f)| - 1$ ODER-Gatter (genauso viele wie die disjunktive Normalform).*

Die Clique-Funktion $CLIQUE(n, s)$ ist auf $\binom{n}{2}$ Variablen definiert, die die Kanten eines ungerichteten Graphen G mit n Knoten repräsentieren. Die Funktion $CLIQUE(n, s)$ nimmt genau dann den Wert 1 an, wenn G eine Clique der Größe s aufweist. Wir zeigen, dass diese Funktion vereinigungsfrei ist. Da die Funktion $CLIQUE(n, s)$ $\binom{n}{s}$ Primimplikanten hat, erhalten wir damit das folgende Korollar.

Korollar 5.5. *Multilineare Schaltkreise für $CLIQUE(n, s)$ benötigen $\binom{n}{s} - 1$ ODER-Gatter.*

Weil nichtdeterministische read-once Branching-Programme von multilinearen Schaltkreisen simuliert werden können, verbessert Korollar 5.5 die untere Schranke von $\exp(\Omega(\min(s, n-s)))$ aus [6] für nichtdeterministische read-once Branching-Programme für $CLIQUE(n, s)$.

Unser nächstes Resultat macht deutlich, dass die Eigenschaft der Vereinigungsfreiheit nicht ausreicht, um gute untere Schranken für allgemeine monotone Schaltkreise zu beweisen. Nach Korollar 5.5 benötigt ein multilinearer Schaltkreis für $CLIQUE(n, n-1)$ $n-1$ ODER-Gatter. Für diese Funktion zeigen wir die folgende obere Schranke für allgemeine monotone Schaltkreise.

Satz 6.1. *Die Funktion $CLIQUE(n, n-1)$ kann durch eine monotone Formel mit $O(\log n)$ ODER-Gattern berechnet werden.*

Dies ist die erste nicht-triviale obere Schranke für die monotone Komplexität der Clique-Funktion. Die einzige andere obere Schranke, die wir kennen, ist in [46] und macht lediglich über die nicht-monotone Komplexität eine Aussage.

Ein Schaltkreis hat *alternierende Tiefe* d, wenn d die größte Zahl von Blöcken aus ODER-Gattern und Blöcken aus UND-Gattern auf Wegen zwischen den Eingängen und Ausgängen des Schaltkreises ist. Ein Σ_d-Schaltkreis (bzw. Π_d-Schaltkreis) ist ein Schaltkreis mit alternierender Tiefe d und einem ODER-Gatter (bzw. UND-Gatter) am Ausgang. Die *Polynom-Funktion* $POLY(q, s)$ wurde von Andreev [2] eingeführt. Sie ist auf q^2 Variablen definiert. Wir beweisen Unkomprimierbarkeit auch für monotone Σ_4-Schaltkreise, die gewisse Funktionen berechnen, und erhalten das folgende Korollar.

Korollar 7.7. *Wenn $s \leq \sqrt{q}/2$, dann muss jeder monotone Σ_4-Schaltkreis für $POLY(q, s)$ mindestens $q^s - 1$ ODER-Gatter haben (genauso viele wie die disjunktive Normalform dieser Funktion).*

Unsere Konstruktion im Beweis von Satz 6.1 liefert einen Π_3-Schaltkreis. Daher legt Korollar 7.7 nahe, dass es schwieriger ist, eine obere Schranke für solche Polynom-Funktionen zu beweisen, als es für die Clique-Funktion ist. Es ist nicht einmal klar, ob diese schwierigen Polynom-Funktionen durch uneingeschränkte Schaltkreise berechnet werden können, die kleiner als die disjunktiven Normalformen sind.

Im Folgenden beschreiben wir den Aufbau dieser Arbeit.

In Kapitel 1 führen wir in das Themengebiet ein und motivieren die Fragestellung.

In Kapitel 2 stellen wir Turing-Maschinen und Boolesche Schaltkreise als Rechenmodelle vor. Wir erläutern den Zusammenhang zwischen der Zeitkomplexität von Turing-Maschinen und Schaltkreiskomplexität. Wir zeigen, dass eine superpolynomiale untere Schranke für die Schaltkreiskomplexität einer Funktion in NP $P \neq NP$ impliziert. Diese Tatsache ist eine wichtige Motivation für die Untersuchung von Schaltkreiskomplexität.

In Kapitel 3 behandeln wir monotone Komplexität. Wir definieren mehrere monotone Boolesche Funktionen, die im weiteren Verlauf der Arbeit von Bedeutung sein werden. Wir führen Slice-Funktionen ein und beweisen, dass Negationen unwichtig für ihre Berechnung sind. Diese Tatsache macht die Untersuchung monotoner Komplexität besonders interessant. Danach stellen wir die Approximationsmethode vor, die zur Zeit die einzige Beweismethode für superpolynomiale monotone untere Schranken ist. Wir wenden die Approximationsmethode auf die Clique-Funktion und die Polynom-Funktion an. Die Resultate dieser Arbeit beziehen sich hauptsächlich auf diese beiden Funktionen.

In Kapitel 4 stellen wir Branching-Programme und geordnete binäre Entscheidungsdiagramme vor. Während allgemeine Branching-Programme von theoretischem Interesse sind, haben auch die umfangreichen praktischen Anwendungen von geordneten binären Entscheidungsdiagrammen zu reichlich Forschung in diesem Bereich geführt. Wir führen multilineare Schaltkreise als eine Verallgemeinerung von nichtdeterministischen read-once Branching-Programmen ein.

In Kapitel 5 beweisen wir die Vereinigungsfreiheit der Clique-Funktion und der Polynom-Funktion (welche lediglich für geeignete Parameter vereinigungsfrei ist). Im Anschluss beweisen wir die untere Schranke für multilineare Schaltkreise vereinigungsfreier Funktionen (Satz 5.4).

In Kapitel 6 beweisen wir die obere Schranke für $CLIQUE\,(n, n-1)$ (Satz 6.1) mit Hilfe fehlerkorrigierender Codes. Wir weiten dieses Resultat auch auf andere Clique-Funktionen aus.

In Kapitel 7 beweisen wir die untere Schranke für monotone Σ_4-Schaltkreise und erhalten Korollar 7.7.

In Kapitel 8 führen wir Pseudo-Slice-Funktionen ein. Wir zeigen, dass die in dieser Arbeit vorgestellten Beweismethoden für untere Schranken auch auf gewisse Pseudo-Slice-Funktionen anwendbar sind. Unsere Beweise der unteren Schranken für multilineare Schaltkreise und für monotone Σ_4-Schaltkreise können mit wenig Aufwand an Pseudo-Slice-Funktionen angepasst werden.

Die Resultate dieser Arbeit sind in [22, 21] veröffentlicht.

Acknowledgements

I am very grateful to Stasys Jukna for many long, helpful and inspiring discussions. He taught me mathematical writing skills and helped me present the material of this thesis in a readable manner. I would also like to thank Georg Schnitger for providing support during my entire studies in Frankfurt and for making this thesis possible.

Last but not least, I would like to thank my parents for their constant support.

Contents

Chapter 1

Introduction

The circuit complexity of a Boolean function is the minimal number of gates of a circuit that computes the function. There are different reasons for studying circuit complexity. One motivation is to investigate the difficulty of recognizing languages with general computers, e.g. the time complexity in the Turing machine model. At first sight, Boolean circuits and Turing machines may seem to be unrelated. However, a Turing machine can be simulated by a series of Boolean circuits, and time and space efficiency of the Turing machine translates to succinctness of the circuits. As a result, it is possible to prove a lower bound on the time complexity of a language by proving a lower bound on the circuit complexity of an associated Boolean function. Thus, circuit complexity offers an approach to the $P = NP$ question and has therefore received much attention. However, the best known lower bound on the complexity of general circuits for functions in NP is only linear in the number of variables, which does not allow us to conclude any properties of Turing machines.

Another motivation for studying circuit complexity comes from hardware design. Of course circuit complexity lower bounds are of interest when looking for an efficient circuit design to be implemented in silicon. But perhaps even more important is knowledge about efficient data structures for representing Boolean functions in computer memory. Boolean functions need to be stored in computers to evaluate and compare them for the purpose of hardware verification. For this restricted circuit models are used that offer efficient algorithms for handling them. Ordered binary decision diagrams were introduced as the first such circuit model, and many other kinds of "decision diagrams" were proposed. Binary decision diagrams are being applied increasingly outside of hardware design. The importance of such data structures provides motivation for studying the complexity of restricted circuit models: we would like to know which data structures are suited for a certain application.

Until now no superpolynomial lower bounds for unrestricted Boolean circuits are known. However, there has been considerable success in proving superpolynomial lower bounds for restricted circuit models such as monotone circuits, which

only use AND and OR gates, but no NOT gates. The method of approximations yields superpolynomial lower bounds for monotone circuits of a number of functions in NP. Also, it is known that negation is almost powerless for so-called slice functions. A superpolynomial lower bound on the *monotone* complexity of a slice function implies a lower bound of the same order on its non-monotone complexity. Unfortunately, the currently available arguments for proving monotone lower bounds seem to be incapable of yielding sufficient lower bounds for slice functions. Therefore it is justified to seek new methods for proving monotone lower bounds.

In this thesis we make some steps in this direction. We study some circuit models that are more restricted than monotone circuits. We prove exponential lower bounds for these restricted circuit classes that are optimal. They state that the circuits require exactly as many OR gates as the DNFs of the considered functions. This means that by using these circuit types instead of DNFs, we cannot even save a single OR gate! In other words, the DNFs are *incompressible* when we restrict ourselves to the respective circuit classes. We regard this incompressibility as a remarkable property of the considered functions. However, we also give an upper bound that shows that some of these DNFs are still highly compressible in the case of general monotone circuits.

We introduce pseudoslice functions, which are similar to slice functions. Since no lower bound arguments applicable to slice functions are known, we suggest to study the complexity of pseudoslice functions. The lower bound arguments that we propose in this thesis also work for pseudoslice functions.

1.1 Organization of the Thesis

In Chapter 2 we introduce Turing machines and Boolean circuits as computational models. We discuss the relationship between the time complexity of Turing machines and circuit complexity. We show that a superpolynomial lower bound on the circuit complexity of a function in NP implies $P \neq NP$. This fact motivates much of the research in circuit complexity.

In Chapter 3 we discuss monotone complexity. We define several monotone functions that will be of importance in the further course of the thesis. We introduce slice functions and prove that negation is powerless for slice functions. This fact makes the study of monotone complexity particularly interesting. Finally, we present the method of approximations, which is the only method currently available for proving superpolynomial lower bounds on monotone complexity. We apply the method of approximations to the clique function and the so-called polynomial function. The results that we present in this thesis apply mainly to these two functions.

In Chapter 4 we introduce branching programs and ordered binary decision diagrams. While general branching programs are of theoretical interest, the ex-

tensive applications of ordered binary decision diagrams have motivated a lot of research in this area. We introduce multilinear circuits as a generalization of ordered binary decision diagrams and nondeterministic read-once branching programs. A Boolean circuit is multilinear if the inputs to each of its AND gates are computed from disjoint sets of variables.

In Chapter 5 we introduce union-free functions. We identify two prominent union-free functions: the clique function and the polynomial function (which is union-free for suitable parameters). We prove an optimal lower bound for multilinear circuits of union-free functions. This bound states that multilinear circuits for union-free functions need just as many OR gates as the respective DNFs of these functions, i.e. they are incompressible. Thus, multilinear circuits are inefficient for union-free functions, although they are adequate for many other functions.

In Chapter 6 we show that our lower bounds for multilinear circuits cannot be extended to unrestricted monotone circuits. We prove that cliques of size $n-1$ in a n-vertex graph can be detected by monotone formulas with $O(\log n)$ OR gates. The DNF of this clique function has $n-1$ OR gates. Since the clique function is union-free, multilinear circuits for this clique function are incompressible and require $n-1$ OR gates as well. Hence, general monotone formulas for this clique function can be much more efficient than multilinear circuits. This is the first non-trivial upper bound on the monotone complexity of the clique function. By exploiting that this particular clique function is a projection of practically all clique functions, we are able to show that general monotone circuits require less OR gates than DNFs for clique functions in general. The formulas we construct for proving this upper bound are Π_3-formulas, i.e. they are conjunctions of disjunctions of monoms.

In Chapter 7 we prove lower bounds for monotone circuits of bounded depth. We show that monotone Σ_4-circuits for a certain class of polynomial functions are incompressible, i.e. they require at least as many OR gates as the DNFs of the respective functions. The class of Σ_4-circuits includes the Π_3-formulas, for which we proved the upper bound in the previous section. This means that the polynomial functions studied in this section are in a certain sense harder to compute than clique functions, whose monotone Π_3-formulas are compressible. We still do not know any non-trivial upper bound for the polynomial function.

In Chapter 8 we introduce pseudoslice functions. We prove that our lower bounds for multilinear circuits and monotone Σ_4-circuits also hold for certain pseudoslice functions. The proofs we gave for our lower bounds can easily be adapted to make them work for pseudoslices.

1.2 Contributions of the Thesis

The main contributions of this thesis are:

1. An *optimal* lower bound for multilinear circuits computing the clique function (Theorems 5.4, 8.4 and Corollary 5.5)

2. An upper bound for the clique function (Theorem 6.1)

3. An *optimal* lower bound for monotone bounded depth circuits computing the polynomial function (Theorem 7.6 and Corollary 7.7)

These results are published in [22, 21].

Chapter 2

Turing Machines and Circuits

In this chapter we introduce Turing machines and Boolean circuits as computational models. We discuss the relationship between the time complexity of Turing machines and circuit complexity. We show that a superpolynomial lower bound on the circuit complexity of a function in NP implies $P \neq NP$.

2.1 Turing Machines

The Turing machine is the most common model of a computer. A Turing machine is controlled by an automaton with a finite number of states. The Turing machine uses a tape as memory. The tape is divided into infinitely many cells which can each store one symbol of a finite alphabet. Here way adopt the convention that the tape is infinite in both directions. A tape head can read or write one cell at a time. In one time step, a Turing machine reads a symbol from the tape, writes a symbol on the tape, decides whether to move left or right on the tape and decides which state to assume next. Formally, a Turing machine is defined by a 7-tuple that determines its behavior.

Definition 2.1. A Turing machine is a 7-tuple

$$(Q, \Gamma, B, q_0, \delta, q_A, q_R)$$

where

- Q is the finite set of states,
- Γ is the finite alphabet of symbols that can be written on the tape,
- $B \notin \Gamma$ is the blank symbol,
- $q_0 \in Q$ is the initial state,
- $\delta : Q \times \{\Gamma \cup \{B\}\} \mapsto Q \times \Gamma \times \{L, R\}$ is the transition function,

- $q_A \in Q$ is the accepting state,
- $q_R \in Q$ is the rejecting state.

Before the computation starts, the tape holds a finite string in Γ^* which is the input to the Turing machine. All tape cells that do not hold the input string contain the blank symbol B. The tape head is positioned over the leftmost symbol of the input string. The Turing machine is in the initial state q_0.

In each computation step the transition function δ determines the behavior of the Turing machine. The transition function is evaluated with the current state and the symbol currently read by the tape head as arguments. The transition function then yields the next state, the symbol to be written on the tape and the direction in which the tape head moves. If the next state is q_A, the Turing machine halts and *accepts*. If the next state is q_R, the Turing machine halts and *rejects*. If the next state is neither q_A nor q_R, the computation continues.

The set of input strings for which a Turing machine M accepts is called the language that M recognizes. The recognition of a language is a very basic kind of computational task. Since there are only two possible outcomes — accept and reject — we call language recognition a decision problem. It is a common assumption that all languages that are recognizable by some physical device are recognizable by a Turing machine. This conjecture is known as Church's hypothesis. Church's hypothesis is backed up by the fact that many more general computational models such as machines with several tapes or random access to their memory can be simulated by Turing machines as defined above.

The resources that may be limited when operating a Turing machine typically are time and space. In this chapter we will only be concerned with the time required by a Turing machine. We return to the problem of limited space in Section 4.1 when we discuss branching programs.

Definition 2.2. For a function $T(n)$, a Turing machine M is $T(n)$ *time-bounded* if the number of steps executed by M before halting is bounded by $T(n)$ for all input strings of length n.

We classify languages according to the time needed by the Turing machines that recognize them. The *complexity class* $DTIME(T(n))$ consists of all languages that are recognizable by $T(n)$ time-bounded Turing machines. An important complexity class is the class

$$P = \bigcup_{k=1}^{\infty} DTIME\left(n^k\right) .$$

The class P consists of the languages that are recognizable in polynomial time. The class P can be regarded as the class of problems that are decidable "efficiently". If a problem does not belong to P, the time needed to decide it is very likely to be prohibitive for many applications.

If we are only studying which languages are recognizable at all, it does not matter whether we choose the Turing machine or a more general machine as our computational model. All known computer models can be simulated by Turing machines. The choice of the computational model does matter however when we are studying the computational complexity of languages. Some computer models have features that allow them to perform computations in considerably less time steps than are required by a Turing machine as defined above. For example, a random access machine or a Turing machine with several tapes can be more efficient because these machines can access their memory easier. However, we note that these more efficient computer models can be simulated by standard Turing machines in time which is polynomial in the time required by the superior machine. Thus, the notion of polynomial time computability is to a large extent independent of the underlying computational model.

In order to define the complexity class NP, we now introduce nondeterministic Turing machines. Turing machines as defined above are called deterministic because the sequence of machine configurations is completely determined by the transition function. Let $\mathcal{P}(A)$ denote the power set of a set A.

Definition 2.3. A *nondeterministic Turing machine* is defined by the 6-tuple

$$(Q, \Gamma, B, q_0, \delta, q_A, q_R)$$

where

- Q, Γ, B, q_0, q_A and q_R have the same meaning as in the case of deterministic Turing machines,
- $\delta : Q \times \{\Gamma \cup \{B\}\} \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function.

The transition function of a deterministic Turing machine yields a tuple containing the next state, the tape symbol to write and the direction of the tape head. In contrast, the transition function of a nondeterministic Turing machine yields a set of such tuples. If this set contains more than one tuple, the Turing machine can choose its transition nondeterministically. Thus, for a nondeterministic Turing machine there typically are several sequences of machine configurations that are possible for an input string. A nondeterministic Turing machine accepts an input string if there is some permitted sequence of computations that ends in the accepting state. The set of input strings that a nondeterministic Turing machine M accepts is called the language that M recognizes.

Definition 2.4. A nondeterministic Turing machine M is $T(n)$ time-bounded if the number of steps executed by M before halting is bounded by $T(n)$ for all input strings of length n.

The complexity class $NTIME\left(T\left(n\right)\right)$ consists of all languages that are recognizable by $T\left(n\right)$ time-bounded nondeterministic Turing machines.

The complexity class NP is defined as

$$NP = \bigcup_{k=1}^{\infty} NTIME\left(n^k\right) .$$

The class NP can be regarded as the class of problems that are verifiable in polynomial time. For every string accepted by a polynomial time nondeterministic Turing machine we can list a polynomial number of configurations that prove the acceptance of the input. The class NP receives a lot of attention because many important problems not known to be in P belong to NP. Many of these problems of practical importance are *NP-complete*. It is known that if there exists a polynomial time algorithm for an NP-complete problem, then $P = NP$ and all problems in NP are decidable efficiently. This is the case because every language in NP is *polynomial time reducible* to every NP-complete language. An introduction to NP-completeness and polynomial time reductions together with a list NP-complete problems is presented in [11].

A proof that $P \neq NP$ would yield superpolynomial lower bounds on the computation time needed for all of the numerous NP-complete problems. To show that $P \neq NP$ it is enough to prove a superpolynomial lower bound on the computation time required by some problem in NP. Until now the question whether $P = NP$ has resisted all attacks. Today this problem is regarded as one of the greatest mathematical challenges. One of the objectives of this thesis is to explore some approaches to the $P = NP$ question that the study of Boolean circuits offers.

2.2 Boolean Circuits and Functions

We now turn to the Boolean circuit as a computational model. Unlike Turing machines, Boolean circuits process an input of fixed size. Boolean circuits operate on values that belong to the set $\mathbb{B} = \{0, 1\}$. A Boolean function is a function that maps a Boolean vector $\mathbb{B}^n$ to $\mathbb{B}$. While Turing machines serve as a high-level model of a computer, a Boolean circuit can be regarded as a low-level model of a hardware component. Boolean circuits have no notion of computation time. The computation of a Boolean circuit is imagined to be completed instantaneously. A Boolean circuit consists of a collection of *gates*. Gates compute a simple function of their input, which is a tuple of bits in $\mathbb{B}$. In this thesis we will only be concerned with circuits that consist of AND, OR and NOT gates. Sometimes we will also introduce gates that yield one of the constant values 0 and 1 in order to simplify an argument.

A NOT gate computes a negation, i.e. it simply flips its input bit. We denote the negation of x by $\neg x$. An AND gate with inputs $x, y \in \mathbb{B}$ computes the value

$x \wedge y$, called the conjunction of x and y. An OR gate with inputs $x, y \in \mathbb{B}$ computes the value $x \vee y$, called the disjunction of x and y. The operations $\wedge$ and $\vee$ are defined as follows:

x	y	$x \wedge y$	$x \vee y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

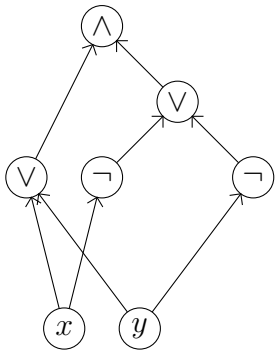
Definition 2.5. A *Boolean circuit* is a directed acyclic graph. There are two disjoint sets of nodes: the inputs and the gates. Every gate is labelled with its type (AND, OR, NOT, 0 or 1).

The number of edges entering a node is called the *fanin* of the vertex. The fanin of an input is 0. The fanin of a gate depends on its type and is between 0 and 2. The number of edges leaving a node is called the *fanout* of the vertex. A circuit whose gates have a fanout of at most 1 is called a *formula*. Nodes with fanout 0 are called output nodes of the circuit.

We now show how to associate a Boolean circuit with n inputs and one output with a Boolean function of n variables that maps $\mathbb{B}^n$ to $\mathbb{B}$. We associate such a function with every node of the circuit. We start with the inputs of the circuit and work ourselves bottom-up through the circuit up to the output. Every input of the circuit corresponds to a variable x_i and is associated with the respective projection $\pi_i : \mathbb{B}^n \mapsto \mathbb{B}$. An AND gate w whose predecessors are u and v with functions f_u and f_v, respectively, is associated with the function

$$f_w(x_1, \ldots, x_n) = f_u(x_1, \ldots, x_n) \wedge f_v(x_1, \ldots, x_n) .$$

We handle gates computing other operations in an analogous manner. The function computed by the circuit is the function associated with the output node.



This circuit with inputs x and y computes their Exclusive OR $x \oplus y = (x \vee y) \wedge (\neg x \vee \neg y)$.

Figure 2.1: Example of a Boolean circuit

A resource that may be limited when dealing with Boolean circuits is the amount of gates that are available. We may not be able to handle circuits that become too large. The size of a circuit is the number of gates used by the circuit.

Definition 2.6. The *combinational complexity* of a Boolean function f is the number of gates used by the circuits for f that use as few gates as possible.

We denote the combinational complexity of a function f by $C(f)$.

It is not a significant restriction if we only allow negations to occur directly at the inputs of a circuit. Then negations can only be applied to variables. A circuit with this restriction is called a *standard circuit*. The following lemma shows that standard circuits are almost as efficient as unrestricted circuits.

Lemma 2.7. *If the function f can be computed by a Boolean circuit with m gates, then f can be computed by a standard circuit with no more than $2m$ gates.*

Proof. Let S be an unrestricted circuit for f. We derive a standard circuit S' for f from S. For every node g of S, which computes some function f_g, we have two nodes g_1 and g_2 in S'. We choose the type of g_1 and g_2 and wire them so that they compute the functions $f_{g_1} = f_g$ and $f_{g_2} = \neg f_g$, respectively.

Let g be an AND gate which is fed from the nodes h and j. The gate g_1 can also be an AND gate, fed from h_1 and j_1. For setting up g_2, we exploit De Morgan's Law $\neg(x \wedge y) = \neg x \vee \neg y$ and make g_2 an OR gate which is fed from h_2 and j_2.

Let g be an OR gate which is fed from the nodes h and j. The gate g_1 can also be an OR gate, fed from h_1 and j_1. For setting up g_2, we exploit De Morgan's Law $\neg(x \vee y) = (\neg x) \wedge (\neg y)$ and make g_2 an AND gate which is fed from h_2 and j_2.

If g is an input node, i.e. a variable, we let g_1 be the same variable and g_2 be a negation gate that negates this variable.

If g is a negation gate fed from node h, we identify g_1 with h_2 and h_2 with h_1. We see now that S' is in fact a standard circuit if we follow these rules for the construction of S'. Through a bottom-up induction it is easy to see that the gates of S' compute the intended functions. □

We now give some preliminaries concerning Boolean functions. A *literal* is a variable or a negated variable. A *monom* is a conjunction of literals. In this thesis we regard monoms also as sets of literals. Therefore, we can compare monoms as we compare sets. For example, for monoms m_1 and m_2 we write $m_1 \subseteq m_2$ if every literal of m_1 also belongs to m_2. An *implicant* of a Boolean function f is a monom that does not evaluate to 1 unless f does. An implicant is a *prime implicant* if no new implicant can be obtained by removing variables or negated variables from the conjunction. For a Boolean function f, we denote the set of its prime implicants by $PI(f)$. We call a Boolean function *k-homogeneous* if each of its prime implicants consists of k literals. A *disjunctive normal form (DNF)* is a

disjunction of monoms. In this thesis we always presume that a DNF is minimal, i.e. the DNF consists of a minimal number of monoms.

2.3 Simulation of Turing Machines by Circuits

Let $L \subseteq \{0,1\}^*$ be a language. We can associate a function $f : \{0,1\}^* \mapsto \{0,1\}$ with L as follows:

$$f(w) := \begin{cases} 1 \text{ for } w \in L \\ 0 \text{ for } w \notin L \end{cases}$$

When we restrict f to arguments of a fixed length n, we obtain the Boolean function $f_n : \mathbb{B}^n \mapsto \mathbb{B}$. We are interested in the combinational complexity of the sequence $f_1, f_2, \ldots$ In this section we show that if L is in P, then the combinational complexity of f_n has an upper bound that is polynomial in n. This translation was first presented in [36]. As a result, a superpolynomial lower bound on the combinational complexity of f_n means that $L \notin P$. If L is in NP, then this would imply $P \neq NP$. Note that restricting the alphabet to $\Gamma = \{0,1\}$ hardly limits the computational abilities of a Turing machine. A larger alphabet can simply be encoded by words in $\{0,1\}^*$.

Definition 2.8. An *oblivious Turing machine* is a Turing machine whose tape head position depends only on the length of the input string and the number of completed computation steps.

To prove that Turing machines can be simulated by Boolean circuits, we first have to show how to transform a given Turing machine into an oblivious Turing machine.

Lemma 2.9. *If a language L is accepted by a $T(n)$ time-bounded Turing machine, then L is accepted by an $O(T^2(n))$ time-bounded oblivious Turing machine.*

Proof. We show how a general $T(n)$ time-bounded Turing machine M_1 can be efficiently simulated by an oblivious Turing machine M_2. Let Γ_1 and Γ_2 be the alphabets of M_1 and M_2, respectively, and let B be the blank symbol of M_1. We set $\Gamma_2 = \Gamma_1 \cup (\Gamma_1 \cup \{B\}) \times \{0,1\}$. M_2 first replaces the leftmost input symbol a by $(a,1)$ and every other input symbol b by $(b,0)$. From now on M_2 only operates with symbols in $(\Gamma_1 \cup \{B\}) \times \{0,1\}$ and its blank symbol. The first component of pairs in $(\Gamma_1 \cup \{B\}) \times \{0,1\}$ represents the symbol of the simulated machine M_1 at the same place on the tape. The second component is 1 if the head of M_1 is over the corresponding tape cell and 0 otherwise.

The machine M_2 keeps sweeping over its tape from left to right and back. M_2 remembers the state of M_1 and the transition of M_1 it has to simulate. When M_2 passes by the tape cell whose symbol has second component 1, it has the opportunity to modify this cell and a neighbor cell in order to simulate the

transition of M_1. Every time it encounters its blank symbol by reaching the left or the right end, it overwrites the blank symbol by (B, x), where x is either 0 or 1, depending on the head position of M_1. We make M_2 move right at the beginning, and it is easy to see that it is possible for M_2 to complete a transition of M_1 after returning to the leftmost non-blank symbol on its tape. Then M_2 can process a second transition and so on.

After the simulation of the t-th transition of M_1, M_2 has $n + 2t$ non-blank symbols on its tape. Hence, M_2 needs $2(n + 2(t-1)) + 1 = 2n + 4t - 3$ steps to simulate the t-th transition of M_1. In the case of a non-trivial language L we have $T(n) \geq n$, so summing up these terms for $t = 1, \ldots, T(n)$ yields a term that is bounded by $O(T^2(n))$. □

The simulation of Turing machines by circuits now follows easily.

Theorem 2.10. *If the language L is accepted by a $T(n)$ time-bounded Turing machine, then the combinational complexity of the function $f_n : \mathbb{B}^n \mapsto \mathbb{B}$ as defined above is bounded by $O(T^2(n))$.*

Proof. According to the Lemma, L is accepted by a $T_M(n) = O(T^2(n))$ time-bounded oblivious Turing machine M. We assume that the tape cells of M are numbered. Let $pos(t, n)$ denote the head position after t steps and $b(t, x, j)$ be the contents of the j-th tape cell after t steps when x is the input string to M. By $q(t, x)$ we denote the state that M is in after t steps for input x.

The machine M can be simulated by successively computing the tape contents after every computation step up to $t = T_M(n)$. To be precise, if δ_1 and δ_2 are the first two projections of the transition function of M, we have

$$\begin{aligned} q(t+1, x) &= \delta_1(q(t, x), b(t, x, pos(t, n))) \\ b(t+1, x, j) &= b(t, x, j) \text{ if } j \neq pos(t, n), \text{ and} \\ b(t+1, x, pos(t, n)) &= \delta_2(q(t, x), b(t, x, pos(t, n))) . \end{aligned}$$

Thus, the tape contents after t steps can be determined by evaluating the transition function t times. Since the transition function is a finite function, it has a constant size circuit. Hence, $T_M(n)$ copies of this circuit for the transition function suffice, and the upper bound on the combinational complexity of f_n follows. □

We conclude that a superpolynomial lower bound on the combinational complexity of f_n means that $L \notin P$. A more sophisticated analysis reveals that a $T(n)$ time-bounded Turing machine can by simulated by circuits of size $O(T(n) \log T(n))$ [26]. An even more careful investigation that also pays attention to the constant factors is due to Schnorr [38].

To prove that $P \neq NP$ it would suffice to show that a series of Boolean functions $f_1, f_2, \ldots$ which decides a language in NP requires circuits with a superpolynomial number of gates. Such a superpolynomial lower bound on the

combinational complexity of a function in NP would be an even more general result than $P \neq NP$. While Turing machines can be simulated by circuits, a series of Boolean functions cannot always be evaluated by a single Turing machine. The reason for this is that a Turing machine has to be based on the same algorithm for all input lengths, whereas the circuits for a series of Boolean functions do not have to be related to each other. This is why Boolean circuits are called a *non-uniform* computational model. Although this observation may make an approach to the $P = NP$ question through circuit complexity seem unnecessarily difficult, Boolean circuits have received a lot of attention because of their relationship to Turing machines. The so-called oracle results [3] have presented evidence that many attempts to prove $P \neq NP$ which are based on Turing machines are determined to fail. Unlike Turing machines, Boolean circuits are amenable to "combinatorial" proof methods. Researchers hoped to be able to use combinatorial methods to derive lower bounds which were not possible with other techniques. However, approaches to the $P = NP$ question via Boolean circuits have not been successful either. Until now, the best lower bound on the combinational complexity of a function in NP is linear. Also for Boolean circuits there have been some efforts to systematically rule out lower bound arguments which could yield the desired lower bound. The theory of *natural proofs* [35] claims that all apparent and manageable combinatorial properties of Boolean functions are incapable of implying a strong lower bound. Schnorr [39] has shown that the unprovability of $P \neq NP$ through a Boolean circuit lower bound is itself unprovable in a certain sense.

In this thesis we will make some steps towards new lower bound arguments for Boolean circuits despite all pessimistic predictions.

Chapter 3

Monotone Circuit Complexity

Proving lower bounds on combinational complexity is very difficult. Until now the best lower bounds are linear in the number of variables. It is natural to first develop lower bound arguments for Boolean circuits with some kind of restriction. *Monotone circuits* are a class of restricted Boolean circuits for which impressive lower bounds have been found.

Definition 3.1. A Boolean circuit is *monotone* if it does not have any NOT gates.

Monotone circuits only use AND and OR gates. They are only capable of computing *monotone functions*, a proper subset of the Boolean functions. A Boolean function f is monotone if the value assumed by f never changes from 1 to 0 when an input variable is changed from 0 to 1. The functions and circuits studied in this thesis are usually monotone.

Definition 3.2. The *monotone complexity* of a monotone function f as the size of the smallest monotone circuits for f.

We denote the monotone complexity of a function f by $C^+(f)$. The minimal DNF of a monotone function is the disjunction of all the prime implicants.

In this chapter we first give some examples of monotone functions. Then we explain the known results concerning the relationship between monotone and combinational complexity. Finally, we present the so-called method of approximations. The method of approximations is, together with its variants, the only currently available technique for proving superpolynomial lower bounds on monotone complexity.

3.1 Examples of Monotone Functions

We will refer to most of these examples of monotone functions in the further course of this thesis.

Example 3.3. The *threshold function* T_k^n of n variables assumes the value 1 if at least k of the inputs are 1.

The following two examples are functions that test properties of graphs.

Example 3.4. The *clique function* $CLIQUE(n,s)$ is a function of $\binom{n}{2}$ variables representing the edges of an undirected n-vertex graph G. An edge belongs to G iff the variable for this edge is set to 1. The function $CLIQUE(n,s)$ assumes the value 1 iff the graph G contains a clique of s vertices.

The clique function is NP-complete. Its monotone complexity is exponential, i.e of the order $\Omega\left(2^{n^c}\right)$ for some constant c. We will prove an exponential lower bound on the monotone complexity in this chapter.

A *perfect matching* of a graph is a subset of its edges such that every vertex of the graph is incident to exactly one of the edges in the subset.

Example 3.5. The *logical permanent* $PM(n)$ is a function of n^2 variables representing the edges of an undirected bipartite graph G with $2n$ vertices. The function $PM(n)$ assumes the value 1 iff the graph G has a perfect matching.

A $\Omega\left(n^{\log n}\right)$ lower bound on the monotone complexity of $PM(n)$ is known [33]. However, the function $PM(n)$ is decidable in polynomial time [16] and thus has polynomial combinational complexity.

The next function we define was introduced by Andreev [2].

Example 3.6. The *polynomial function* $POLY(q,s)$ has q^2 variables corresponding to the points in the grid $GF(q) \times GF(q)$, where q is a prime power. This function accepts a $q \times q$ 0-1 matrix $X = (x_{i,j})$ iff there is a polynomial $f(z)$ of degree at most $s-1$ over $GF(q)$ such that $x_{i,f(i)} = 1$ for all $i \in GF(q)$. These entries $x_{i,f(i)}$ can be regarded as a graph of the polynomial f.

In this chapter we prove an exponential lower bound on the monotone complexity of the function $POLY$, which is in NP.

3.2 The Relationship between Monotone and Combinational Complexity

While every monotone function can be computed by a monotone circuit, a non-monotone circuit may be able to compute a monotone function more efficiently than a monotone circuit. In other words, the monotone complexity of a function may be larger than its combinational complexity. The logical permanent $PM(n)$ is an example of a function with considerable larger monotone complexity than combinational complexity. A $\Omega\left(n^{\log n}\right)$ lower bound on the monotone complexity of $PM(n)$ is known [33]. On the other hand, this function is decidable in polynomial time [16] and thus has polynomial combinational complexity. Tardos was

able to show that the quotient of monotone and combinational complexity can be exponential by finding a function which exhibits such a gap [42].

Since lower bounds are easier to prove for monotone circuits than for unrestricted circuits, it is interesting to study functions whose monotone complexity is not much larger than their combinational complexity. A class of functions with this property are the *slice functions*.

Definition 3.7. Let f be a Boolean function of n variables. The t-slice function of f is the function $f_t = f \wedge T_t^n \vee T_{t+1}^n$, where T_t^n is the t-th threshold function of n variables.

Let $|x|$ denote the number of ones in the Boolean vector x. Then an equivalent definition of the t-slice f_t of f is

$$f_t(x) = \begin{cases} 0 & \text{for } |x| < t \\ f(x) & \text{for } |x| = t \\ 1 & \text{for } |x| > t \end{cases} .$$

The value of the slice function f_t is only meaningful if its argument has exactly t ones.

In order to show that the monotone and combinational complexity of slice functions are close to each other, we introduce the notion of a pseudo-complement. A monotone function h is a *pseudo-complement* for the variable x when computing f if we can replace $\neg x$ by a subcircuit for h in any standard circuit for f without altering the function f computed by the circuit. It turns out that efficient pseudo-complements can be found when computing slice functions. In the sequel, we define $X = \{x_1, \ldots, x_n\}$ and $X_i = X \setminus \{x_i\}$. Then we can write the threshold function $T_t^{n-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ as $T_t^{n-1}(X_i)$.

Theorem 3.8 ([5]). *When computing the t-slice function $f_t(x_1, \ldots, x_n)$ of $f(x_1, \ldots, x_n)$, the function $T_t^{n-1}(X_i)$ is a pseudo-complement of the variable x_i.*

Proof. We distinguish three cases:

Case 1: $|x| < t$. Then $f_t(x_1, \ldots, x_n) = 0$. We have $T_t^{n-1}(X_i) = 0$. Thus, replacing $\neg x_i$ by $T_t^{n-1}(X_i)$ means replacing $\neg x_i$ by 0 in this case. Since we are dealing with a standard circuit in which negations are only applied to variables, the value computed remains 0 after this replacement is done.

Case 2: $|x| = t$. If $x_i = 0$, then $\neg x_1 = 1$ and $T_t^{n-1}(X_i) = 1$ because exactly t of the variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ are set to 1. If $x_i = 1$, then $\neg x_1 = 0$ and $T_t^{n-1}(X_i) = 0$ because exactly $t-1$ of the variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ are set to 1. Hence, $\neg x_i = T_t^{n-1}(X_i)$ in this case.

Case 3: $|x| > t$. Then $f_t(x_1, \ldots, x_n) = 1$. We have $T_t^{n-1}(X_i) = 1$. Thus, replacing $\neg x_i$ by $T_t^{n-1}(X_i)$ means replacing $\neg x_i$ by 1 in this case. Since we are dealing with a standard circuit, the value computed remains 1 after this replacement is done. □

Since threshold functions can be computed efficiently, Theorem 3.8 implies that unrestricted circuits for slice functions can be simulated efficiently by monotone circuits. We now make more precise how efficient this simulation is. A simple monotone circuit for the threshold function T_k^n can be constructed with $O(nk)$ gates. Hence, we are able to compute the n pseudo-complements given by Theorem 3.8 that we need for a slice function of n variables with $O(n^3)$ gates. Thus, the difference between the monotone complexity and the combinational complexity of a slice function of n variables is at most $O(n^3)$. As a result, a superpolynomial lower bound on the monotone complexity of a slice function yields a superpolynomial lower bound on its combinational complexity. A more careful analysis reveals an even closer relationship between the monotone and the combinational complexity of slice functions. According to Theorem 3.8, the pseudo-complements we need to compute a t-slice function of n variables are the vector $\left(T_t^{n-1}(X_1), \ldots, T_t^{n-1}(X_n)\right)$. The best upper bound for this vector has been given by Valiant [43]. Wegener [45] gives a shorter proof of this upper bound.

Theorem 3.9 ([43]). *The vector $\left(T_t^{n-1}(X_1), \ldots, T_t^{n-1}(X_n)\right)$ can be computed by a monotone circuit with $O\left(n \log^2 n\right)$ gates.*

Corollary 3.10. *A lower bound of $\omega\left(n \log^2 n\right)$ on the monotone complexity of a slice function in n variables implies a lower bound of the same order on the combinational complexity of this slice function.*

Since threshold functions of n variables can be computed by unrestricted circuits with $O(n)$ gates [23], we have $C(f_t) \leq C(f) + O(n)$. So a superlinear lower bound on the combinational complexity of a slice function of a function f implies a lower bound of the same order on the combinational complexity of f.

Corollary 3.11. *Let f be a Boolean function of n variables. A lower bound of $\omega\left(n \log^2 n\right)$ on the monotone complexity of any slice function f_t of f implies a lower bound of the same order on the combinational complexity of f.*

We have shown how to derive a lower bound on combinational complexity from a lower bound on monotone complexity. On the other hand, it is easy to see that a high lower bound on the combinational complexity of a function implies a high lower bound on the complexity of one of its slices. Recall that $f(x) = f_t(x)$ when $|x| = t$. So to recover a function from its slice functions, it is enough to count the number of input bits which are one and then select the appropriate slice function using a multiplexer. Counting and multiplexing can be achieved using $O(n)$ gates if we are dealing with a function of n variables. As a result we obtain

$$C(f) = O\left(n + \sum_{t=0}^{n} C(f_t)\right).$$

We conclude that at least one of the slices f_t of f has a combinational complexity of $C(f_t) = \Omega(C(f)/n)$. In some cases it is possible to single out a particular slice that is known to be "hard".

Theorem 3.12 ([10]). *The $\binom{n}{2}/2$-slice of the function $CLIQUE(n,s)$ is NP-complete.*

3.3 The Method of Approximations

The method of approximations is up to now the only method available for proving superpolynomial bounds on monotone complexity. The method of approximations was introduced by Razborov [32]. Some other methods have been proposed [2, 13, 18], but these are just different variants of the method of approximations. Some efforts have been made to present the method of approximations in a way that is easier to understand [4, 18]. In this section we describe the method of approximations following the older approach of Alon and Boppana [1], thereby allowing a few simplifications. This traditional variant of the method of approximations is also presented in the monographs of Wegener [46] and Dunne [9].

In an ordinary monotone circuit, an AND or OR gate is used to create a new function from the functions that correspond to the inputs to the gate. When applying the method of approximations, we consider the "approximator circuits" we obtain from monotone circuits by replacing ordinary gates by lattice operations. A *lattice* is an algebra with two operations, $\sqcap$ and $\sqcup$ (called "meet" and "join", respectively), that are idempotent, commutative and associative and that satisfy the absorption law ($a \sqcup a \sqcap b = a$). As the elements of our lattice $\mathcal{K}$, we choose a subset of the monotone Boolean functions from $\mathbb{B}^n$ to $\mathbb{B}$. Thus, the functions at the gates of the approximator circuit can often only be approximations of the functions computed at the gates of the original circuit. We identify a Boolean function $f : \mathbb{B}^n \mapsto \mathbb{B}$ with the set $f^{-1}(1)$ of Boolean vectors that it maps to 1. We set $A(f) := \{v \in \mathbb{B}^n \mid f(v) = 1\}$. Hence, the lattice elements we deal with are subsets of $\mathbb{B}^n$. We choose the lattice operations in a way that makes them yield the best possible approximation in a certain sense. For $A, B \in \mathcal{K}$, we define $A \sqcap B$ to be the maximal element of $\{C \in \mathcal{K} \mid C \subseteq A \cap B\}$ with respect to inclusion. We define $A \sqcup B$ to be the minimal element of $\{C \in \mathcal{K} \mid C \supseteq A \cup B\}$. To guarantee that these sets are nonempty, we require the empty set and $\mathbb{B}^n$ to belong to the lattice. Thus, the approximate function $A \sqcap B$ rejects all inputs that the exact function $A \cap B$ rejects. Similarly, the approximate function $A \sqcup B$ accepts all inputs that the exact function $A \cup B$ accepts. The method of approximations relies on an analysis of the errors

$$\delta_{\sqcap}(A,B) = (A \cap B) \setminus (A \sqcap B)$$

and

$$\delta_{\sqcup}(A,B) = (A \sqcup B) \setminus (A \cup B)$$

that are introduced by the lattice operations.

The idea is to choose a lattice that can approximate monotone functions with medium accuracy. This allows us to derive a lower bound on the number of inputs for which the function computed by the approximator circuit differs from the function we would like to actually compute. On the other hand, we are able to find an upper bound on the size of the error introduced by a single lattice operation. From these two bounds we can derive a lower bound on the size of the circuit.

In order to be able to replace ordinary gates by lattice operations, we need to have suitable lattice elements that correspond to variables. Therefore we restrict ourselves to *legitimate lattices*. A legitimate lattice is a lattice $\mathcal{K}$ such that $A(x_1), \ldots, A(x_n) \in \mathcal{K}$. As already mentioned above, we require that $\emptyset \in \mathcal{K}$ and $\mathbb{B}^n \in \mathcal{K}$. We can bound the monotone complexity of a function by its *distance* to a legitimate lattice $\mathcal{K}$. For a monotone function f and a legitimate lattice $\mathcal{K}$, the distance $\varrho(f, \mathcal{K})$ is the minimum t such that there are $A, A_1, B_1, \ldots, A_t, B_t \in \mathcal{K}$ satisfying

$$A \subseteq A(f) \cup \bigcup_{i=1}^{t} \delta_{\sqcup}(A_i, B_i) \tag{3.1}$$

and

$$A(f) \subseteq A \cup \bigcup_{i=1}^{t} \delta_{\sqcap}(A_i, B_i) \ . \tag{3.2}$$

This definition is inspired by the idea of turning monotone circuits into approximator circuits. The pairs $(A_1, B_1), \ldots, (A_t, B_t)$ of sets are thought of as inputs to lattice operations of the approximator circuit, and A is supposed to be the set at the output of the approximator circuit. Equation (3.1) expresses that all Boolean vectors accepted by the function computed by the approximator circuit are accepted by the function computed by the original circuit or have been affected by an error introduced by a lattice operation. Similarly, equation (3.2) reflects that all Boolean vectors rejected by the function computed by the approximator circuit are rejected by the function computed by the original circuit or have been affected by an error. By observing that we can turn every monotone circuit into an approximator circuit by replacing Boolean operations with lattice operations, we can prove the following theorem.

Theorem 3.13 ([32, 1]). *For every monotone function f and every legitimate lattice $\mathcal{K}$, we have $C^+(f) \geq \varrho(f, \mathcal{K})$.*

The proof of this theorem and other proofs omitted in this section are given in the appendix.

The essential effort that has to be made when applying the method of approximations is to choose a lattice that provides for approximations of medium

accuracy. Consider the legitimate lattice $\mathcal{K} = \mathcal{P}(\mathbb{B}^n)$ which contains every function, so approximation is not necessary. Every monotone function has distance 0 to this lattice. As an example of a lattice that only allows very poor approximations, consider the legitimate lattice $\mathcal{K}$ that only has the elements $\emptyset$, $\mathbb{B}^n$ and $A(x_1), \ldots, A(x_n)$. It is easy to see that every monotone function has a distance of at most $n(n-1)$ to $\mathcal{K}$.

3.3.1 A Lower Bound for the Clique Function

For a fixed number r, we say the sets $W_1, \ldots, W_r$ *imply* the set W if the containment $W_i \cap W_j \subseteq W$ holds for all pairs i, j such that $1 \leq i < j \leq r$. We denote this relationship by $W_1, \ldots, W_r \vdash W$. If A is a set of sets and W is a set, we say that A implies W ($A \vdash W$) iff there exist $W_1, \ldots, W_r \in A$ that imply W. We call a set $A \subseteq U$ a *closed subset* of U iff every set $W \in U$ that is implied by A belongs to A.

We are now ready to describe the lattice we choose for proving the lower bound for the clique function. Let V be the set of vertices of the graph on which we detect cliques. We have n variables $x_1, \ldots, x_n$ for each of the possible edges of this graph. For a set A of subsets of V, we define $\lceil A \rceil$ as the set of all graphs on V that contain a clique on some $W \in A$. Each such graph is represented by a characteristic vector in $\mathbb{B}^n$. Thus, we can regard a set $\lceil A \rceil$ as a monotone Boolean function that maps $\mathbb{B}^n$ to $\mathbb{B}$. We choose our lattice as a certain set of such functions $\lceil A \rceil$. First, let $\mathcal{V}(l) = \{W \subseteq V \mid |W| \leq l\}$ be the set of all subsets of V with size at most l. Then we can choose our lattice as

$$\mathcal{K}(r, l) := \{\lceil A \rceil \mid A \text{ is a closed subset of } \mathcal{V}(l)\} .$$

We define the closure A^* of a set A as the smallest subset of $\mathcal{V}(l)$ that contains A and is closed. The lattice operations are given by $\lceil A \rceil \sqcup \lceil B \rceil = \lceil (A \cup B)^* \rceil$ and $\lceil A \rceil \sqcap \lceil B \rceil = \lceil (A \cap B) \rceil$. If the variable x represents the edge that is incident with the vertices u and v, then $A(x) = \lceil \{\{u, v\}\} \rceil$. Since $\{\{u, v\}\}$ is a closed subset of $\mathcal{V}(l)$, we have $A(x) \in \mathcal{K}(r, l)$. Hence, the lattice $\mathcal{K}(r, l)$ is a legitimate lattice.

In order to analyze the approximation properties of our lattice, we need to first prove a combinatorial lemma. This lemma will also be useful for studying other lattices.

We say that a set of sets $\mathcal{F}$ has property $P(r, k)$ if

(i) every set $W \in \mathcal{F}$ has cardinality at most k, and

(ii) there are no (not necessarily distinct) $W, W_1, \ldots, W_r \in \mathcal{F}$ and $U \subsetneq W$ such that $W_i \cap W_j \subseteq U$ for all $1 \leq i < j \leq r$ (so $\mathcal{F} \vdash U$).

By $h(r, k)$ we denote the maximum possible cardinality of a set $\mathcal{F}$ that has property $P(r, k)$.

Lemma 3.14. *For all $r \geq 2$ and $k \geq 0$, we have $h(r, k) \leq (r-1)^k$.*

Corollary 3.15. *Let A be a closed subset of $\mathcal{V}(l)$. Then for all k there are at most $(r-1)^k$ minimal elements (with respect to containment) of A of cardinality at most k.*

Proof. It is easy to see that all minimal elements of A with cardinality at most k form a set with property $P(r,k)$. The corollary then follows directly from Lemma 3.14. □

In order to compare the difference between the output of the approximator circuit and the original circuit with the errors introduced by individual lattice operations, we restrict our attention to inputs to the circuits that we call test graphs. *Positive test graphs* are inputs to the circuit that the function we are studying maps to 1. *Negative test graphs* are inputs that are mapped to 0. Test graphs are Boolean vectors in $\mathbb{B}^n$. A $\sqcap$-operation may introduce new errors that affect positive test graphs. An approximating lattice element resulting from a $\sqcap$-operation may map some positive test graphs to 0 even if these test graphs are mapped to 1 by both operands. Conversely, we study the approximating capabilities of the $\sqcup$-operation for negative test graphs. We attach a probability distribution to our negative test graphs, whereas for our purposes this is not necessary for positive test graphs. We denote the set of positive test graphs by E_+. Concerning the positive test graphs $E_+ \subseteq \mathbb{B}^n$, we are interested in finding an upper bound for the maximum error

$$\delta_+(r,l) := \max_{A,B \in \mathcal{K}(r,l)} |\delta_\sqcap(A,B) \cap E_+|$$

which can be traced to a single lattice operation.

If we are detecting cliques of size s, we choose as positive test graphs E_+ simply all cliques of size s. Further we assume that we are dealing with graphs with m vertices. Using the previous corollary, this leads to the following bound on $\delta_+(r,l)$.

Lemma 3.16. *We have $\delta_+(r,l) \leq (r-1)^{2l} \cdot \binom{m-l-1}{s-l-1}$.*

Proof. By definition, for lattice elements $M, N \in \mathcal{K}(r,l)$ we have

$$\begin{aligned} \delta_\sqcap(M,N) &= (\lceil A \rceil \cap \lceil B \rceil) \setminus (\lceil A \rceil \sqcap \lceil B \rceil) \\ &= (\lceil A \rceil \cap \lceil B \rceil) \setminus (\lceil A \cap B \rceil) \end{aligned}$$

for closed subsets $A, B \subseteq \mathcal{V}(l)$. Consider a graph G in $\lceil A \rceil \cap \lceil B \rceil$. There must be minimal elements $W_1 \in A$ and $W_2 \in B$ such that G has cliques on both of the vertex sets W_1 and W_2. If $|W_1 \cup W_2| \leq l$, then G is in $\lceil A \cap B \rceil$, so $G \notin \delta_\sqcap(M,N)$. According to Corollary 3.15, there are at most $(r-1)^l$ minimal elements in each A and B. Hence, there are at most $(r-1)^{2l}$ combinations of such minimal elements $W_1 \in A$ and $W_2 \in B$. We only need to consider such a combination if $|W_1 \cup W_2| \geq l+1$, and in this case at most $\binom{m-l-1}{s-l-1}$ s-cliques (positive test graphs) have cliques on $W_1 \cup W_2$. Altogether, there can be at most $(r-1)^{2l} \cdot \binom{m-l-1}{s-l-1}$ positive test graphs in $\delta_\sqcap(M,N)$. □

Our next step is to study the approximation accuracy of our lattice with negative test graphs. As negative test graphs, we choose $(s-1)$-partite graphs. We attach a probability distribution to the negative test graphs. We generate a $(s-1)$-partite graph randomly by coloring each vertex independently with one of $s-1$ colors, with each choice of color being equally likely. The random variable we will work with is a random function O that assigns to each of the m vertices one of $s-1$ colors. The graph we obtain from such a coloring is constructed by adding an edge between two vertices iff the colors assigned to these vertices are distinct. We denote this graph by $G(O)$. It is easy to see that such a graph does not contain any s-clique.

When considering negative test graphs, we are interested in the maximum probability

$$\delta_{-}(r,l) := \max_{A,B \in \mathcal{K}(r,l)} \mathbb{P}\left[G(O) \in \delta_{\sqcup}(A,B)\right]$$

that the randomly chosen negative test graph $G(O)$ is affected by an error that is introduced by some individual $\sqcup$-operation. In order to study the properties of the sets $\delta_{\sqcup}(\lceil A \rceil, \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus (\lceil A \rceil \cup \lceil B \rceil)$, we imagine that the closure $(A \cup B)^*$ is constructed from $A \cup B$ by successively adding new sets $W_1, \ldots, W_p$. Such a set W_i is an arbitrary set such that $A \cup B \cup \{W_1, \ldots, W_{i-1}\} \vdash W_i$. In order to bound the probability $\delta_{-}(r,l)$, we study how random colorings behave with respect to such implications of new sets W_i. We call a set W of vertices *properly colored* by a random coloring O if each vertex in W has a different color.

Lemma 3.17. *Let $A \subseteq \mathcal{V}(l)$, $A \vdash W$ and O be a random $(s-1)$-coloring of all vertices. Then*

$$\mathbb{P}\left[W \text{ is properly colored by } O \text{ and no set in } A \text{ is properly colored by } O\right] \leq \left(1 - \frac{(s-1)(s-2)\cdots(s-l)}{(s-1)^l}\right)^r .$$

Using the previous lemma, we can give a bound on $\delta_{-}(r,l)$.

Lemma 3.18.

$$\delta_{-}(r,l) \leq m^l \left(1 - \frac{(s-1)(s-2)\cdots(s-l)}{(s-1)^l}\right)^r .$$

Proof. We have

$$\delta_{\sqcup}(\lceil A \rceil, \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus (\lceil A \rceil \cup \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus \lceil A \cup B \rceil .$$

Thus, a graph in $\delta_{\sqcup}(\lceil A \rceil, \lceil B \rceil)$ has a clique on a vertex set that is in $(A \cup B)^*$ but not in $A \cup B$. Therefore we are going to study the probability that a random

graph $G(O)$ has a clique on some set in $(A \cup B)^*$ but not on any set in $A \cup B$. We imagine that the closure $(A \cup B)^*$ is constructed from $A \cup B$ by successively adding new sets $W_1, \ldots, W_p$, where $A \cup B \cup \{W_1, \ldots, W_{i-1}\} \vdash W_i$. A random graph $G(O)$ is in $\lceil C \rceil$ iff some vertex set in C is properly colored by the coloring O. Using Lemma 3.17, we observe that the probability that a random graph $G(O)$ has a clique on some set in $A \cup B \cup \{W_1, \ldots, W_i\}$ but not on any set in $A \cup B \cup \{W_1, \ldots, W_{i-1}\}$ is at most

$$\left(1 - \frac{(s-1)(s-2)\cdots(s-l)}{(s-1)^l}\right)^r .$$

The number p of sets added is at most $|\mathcal{V}(l)| \leq m^l$. The bound on $\delta_-(r,l)$ follows. □

Having derived upper bounds on $\delta_+(r,l)$ and $\delta_-(r,l)$, we are now ready to prove the lower bound on the monotone complexity of the clique function.

Theorem 3.19. *For $s \geq 16$, we have*

$$C^+(CLIQUE(m,s)) \geq \frac{1}{3}\left(\frac{m}{32s^2\log^2 m}\right)^{\sqrt{s}} .$$

Proof. We choose $l = \lceil\sqrt{s}\rceil$ and $r = \lceil 4\lceil\sqrt{s}\rceil \log m\rceil$. Our goal is to show

$$\varrho(CLIQUE(m,s), \mathcal{K}(r,l)) \geq \frac{1}{3}\left(\frac{m}{32s^2\log^2 m}\right)^{\sqrt{s}} .$$

The lower bound for $C^+(CLIQUE(m,s))$ then follows with Theorem 3.13. Let t be the distance $\varrho(CLIQUE(m,s), \mathcal{K}(r,l))$. We refer to the lattice elements $A, A_1, B_1, \ldots, A_t, B_t$ used in the equations (3.1) and (3.2).

Case 1: The lattice element $A = \lceil C \rceil$ is not the empty set. We investigate the probability that a random graph $G(O)$ is in A. Let W be some vertex set in C. There are $(s-1)^{|W|}$ equally likely possible combinations of colors that a random coloring O can assign to the vertices of W. If O properly colors the vertices of W, then $G(O)$ certainly is in A. There are $(s-1)(s-2)\cdots(s-|W|)$ proper colorings of the vertices in W. With $|W| \leq l$ these observations give us

$$\begin{aligned}
\mathbb{P}[G(O) \in A] &\geq \frac{(s-1)(s-2)\cdots(s-|W|)}{(s-1)^{|W|}} \\
&\geq \frac{(s-1)(s-2)\cdots(s-l)}{(s-1)^l} \\
&= \frac{(s-1)(s-2)\cdots(s-\lceil\sqrt{s}\rceil)}{(s-1)^{\lceil\sqrt{s}\rceil}} .
\end{aligned}$$

We analyze this term further. We have

$$\begin{aligned}
\frac{(s-1)(s-2)\cdots(s-\lceil\sqrt{s}\rceil)}{(s-1)^{\lceil\sqrt{s}\rceil}} &= \frac{s-1}{s-1}\cdot\frac{s-1-1}{s-1}\cdots\frac{s-1-(\lceil\sqrt{s}\rceil-1)}{s-1} \\
&= \left(1-\frac{1}{s-1}\right)\left(1-\frac{2}{s-1}\right)\cdots\left(1-\frac{\lceil\sqrt{s}\rceil-1}{s-1}\right) \\
&\geq 1-\frac{1}{s-1}\sum_{i=1}^{\lceil\sqrt{s}\rceil-1} i \\
&= 1-\frac{\lceil\sqrt{s}\rceil(\lceil\sqrt{s}\rceil-1)}{2(s-1)} \\
&\geq 1-\frac{s+\sqrt{s}}{2(s-1)}
\end{aligned}$$

Since we are assuming $s \geq 16$, we have $s+\sqrt{s} \leq \frac{4}{3}(s-1)$, which yields $\mathbb{P}[G(O) \in A] \geq 1/3$. From equation (3.1) we conclude that

$$\mathbb{P}[G(O) \in A] \leq \mathbb{P}[G(O) \in A(f)] + t \cdot \delta_-(r,l) .$$

Since the graph $G(O)$ is a negative test graph, $\mathbb{P}[G(O) \in A(f)] = 0$, so $t \geq \mathbb{P}[G(O) \in A]/\delta_-(r,l)$. Using the inequality

$$\frac{(s-1)(s-2)\cdots(s-\lceil\sqrt{s}\rceil)}{(s-1)^{\lceil\sqrt{s}\rceil}} \geq 1/3$$

we just proved above, we can further analyze the bound on $\delta_-(r,l)$ given in Lemma 3.18:

$$\begin{aligned}
\delta_-(r,l) &\leq m^{\lceil\sqrt{s}\rceil}\left(1-\frac{(s-1)(s-2)\cdots(s-\lceil\sqrt{s}\rceil)}{(s-1)^{\lceil\sqrt{s}\rceil}}\right)^{\lceil 4\lceil\sqrt{s}\rceil\log m\rceil} \\
&\leq m^{\lceil\sqrt{s}\rceil}\left(\frac{2}{3}\right)^{\lceil 4\lceil\sqrt{s}\rceil\log m\rceil} \\
&\leq m^{\lceil\sqrt{s}\rceil}m^{-2\lceil\sqrt{s}\rceil} \\
&= m^{-\lceil\sqrt{s}\rceil} .
\end{aligned}$$

This directly leads to the desired bound

$$t \geq \frac{\mathbb{P}[G(O) \in A]}{\delta_-(r,l)} \geq \frac{1}{3}m^{\lceil\sqrt{s}\rceil} \geq \frac{1}{3}\left(\frac{m}{32s^2\log^2 m}\right)^{\sqrt{s}} ,$$

which finishes case 1.

Case 2: The lattice element A is the empty set. For this case we turn to our positive test graphs. From equation (3.2) we conclude that

$$|E_+| \leq |A \cap E_+| + t \cdot |\delta_+(r,l)| \ .$$

Since A is empty, $|A \cap E_+| = 0$, so $t \geq |E_+| / |\delta_+(r,l)|$. There are $\binom{m}{s}$ possible s-cliques in a graph with m vertices, so $|E_+| = \binom{m}{s}$. Using the bound on $\delta_+(r,l)$ from Lemma 3.16, we obtain

$$\begin{aligned} t &\geq \frac{\binom{m}{s}}{(r-1)^{2l} \cdot \binom{m-l-1}{s-l-1}} \\ &= \frac{m!\,(s-l-1)!}{(m-l-1)!s!\,(r-1)^{2l}} \\ &\geq \frac{m^{l+1}}{s^{l+1}\,(r-1)^{2l}} \\ &\geq \left(\frac{m}{s\,(r-1)^2}\right)^{\sqrt{s}} . \end{aligned}$$

Since $s \geq 16$, we have $\lceil\sqrt{s}\rceil^2 \leq 2s$, so $(r-1)^2 \leq 32s\log^2 m$. This gives us

$$t \geq \left(\frac{m}{32s^2\log^2 m}\right)^{\sqrt{s}} .$$

This finishes case 2 and the proof of the theorem. □

3.3.2 A Lower Bound for the Polynomial Function

Now we show how to apply the method of approximations to the polynomial function $POLY(q,s)$. The polynomial function can be handled in a similar way as the clique function. We identify the variables of the polynomial function with elements of $GF(q) \times GF(q)$. We choose a lattice that consists of functions that are disjunctions of sufficiently short monoms. Let

$$\mathcal{E}(l) := \{F \subseteq GF(q) \times GF(q) \mid |F| \leq l\}$$

be the set of monoms of length at most l. For a subset A of $\mathcal{E}(l)$, we define $\lceil A\rceil$ as the set of all $b \in \mathbb{B}^{q^2}$ for which there is some $F \in A$ such that all entries of b that correspond to elements of F are 1. When regarded as a Boolean function, $\lceil A\rceil$ is the disjunction of all monoms in A. For some fixed number r, we take the definition of the implication $W_1, \ldots, W_r \vdash W$ from our treatment of the clique function. We also use the same notion of a closed subset. Now we are ready to define the lattice that we use for the polynomial function:

$$\mathcal{K}(q,r,l) := \{\lceil A\rceil \mid A \text{ is a closed subset of } \mathcal{E}(l)\}$$

We define the closure A^* of a set A as the smallest subset of $\mathcal{E}(l)$ that contains A and is closed. The lattice operations are given by $\lceil A \rceil \sqcup \lceil B \rceil = \lceil (A \cup B)^* \rceil$ and $\lceil A \rceil \sqcap \lceil B \rceil = \lceil (A \cap B) \rceil$. If the variable x represents the element $(g_1, g_2) \in GF(q) \times GF(q)$, then $A(x) = \lceil \{\{(g_1, g_2)\}\} \rceil$. Since $\{\{(g_1, g_2)\}\}$ is a closed subset of $\mathcal{E}(l)$, we have $A(x) \in \mathcal{K}(q, r, l)$. Hence, the lattice $\mathcal{K}(q, r, l)$ is a legitimate lattice.

Also for the polynomial function we have to choose a set of positive test graphs E_+ and a distribution of a random negative test graph G. Since the argument of the polynomial function, a $q \times q$ 0-1 matrix, can be regarded as the representation of a graph, it is justified to speak of test graphs in this context too. We proceed by proving upper bounds on

$$\delta_+(q, r, l) := \max_{A,B \in \mathcal{K}(q,r,l)} |\delta_\sqcap(A, B) \cap E_+|$$

and

$$\delta_-(q, r, l) := \max_{A,B \in \mathcal{K}(q,r,l)} \mathbb{P}[G \in \delta_\sqcup(A, B)] .$$

As the positive test graphs E_+, we choose all Boolean matrices $(b_{i,j})$ for which there is a polynomial f of degree at most $s-1$ such that $b_{i,f(i)} = 1$ iff $f(i) = i$.

Lemma 3.20. *If $r \leq q/3 + 1$, then*

$$\delta_+(q, r, l) \leq 3q^{s - \lceil (l+1)/2 \rceil} (r-1)^{\lceil (l+1)/2 \rceil} .$$

The next step is to find a bound on $\delta_-(q, r, l)$. As negative test graph G, we choose the Boolean vector we obtain by setting every entry independently to 1 with probability $1 - \varepsilon$. Note that this random test graph may well be accepted by the function $POLY(q, s)$. We can deal with this by bounding the probability that the random vector G is accepted. For every polynomial f, the probability that all the entries corresponding to some $(g, f(g))$ for $g \in GF(q)$ are set to 1 is $(1-\varepsilon)^{-q}$ since there are q such entries. Since there are q^s polynomials of degree at most $s-1$, we obtain

$$\mathbb{P}[G \in A(f)] \leq q^s (1-\varepsilon)^{-q} \leq q^s e^{-\varepsilon q} .$$

We choose $\varepsilon = (s \ln q + \ln 2)/q$. This gives us $\mathbb{P}[G \in A(f)] \leq 1/2$.

Using our choice of negative test graph, we obtain the following bound on $\delta_-(q, r, l)$.

Lemma 3.21. *We have*

$$\delta_-(q, r, l) \leq q^{2l} (\varepsilon l)^r .$$

We are now ready to give the lower bound for the polynomial function.

Theorem 3.22. *For $s \leq \frac{1}{12}\sqrt{q/\ln q}$, we have*

$$C^+(POLY(q, s)) = q^{\Omega(s)} .$$

Proof. We choose $l = s$ and $r = \lceil 4s \ln q \rceil$. This makes $r - 1 \leq 4s \ln q \leq q/3$. Our goal is to show

$$\varrho\left(POLY\left(q,s\right), \mathcal{K}\left(q,r,l\right)\right) = q^{\Omega(s)}.$$

The lower bound for $C^+\left(POLY\left(q,s\right)\right)$ then follows with Theorem 3.13. Let t be the distance $\varrho\left(POLY\left(q,s\right), \mathcal{K}\left(q,r,l\right)\right)$. We refer to the lattice elements $A, A_1, B_1, \ldots, A_t, B_t$ listed in equations (3.1) and (3.2).

Case 1: $A \neq \mathbb{B}^{q^2}$. From equation (3.2), we know that

$$t \geq \frac{|E_+| - |A \cap E_+|}{\delta_+\left(q,r,l\right)}. \tag{3.3}$$

We would like to estimate $|A \cap E_+|$. Let $A = \lceil C \rceil$. Since $A \neq \mathbb{B}^{q^2}$, every set $F \in C$ has at least one element. According to Corollary 3.15, the set C has at most $(r-1)^k$ minimal elements of cardinality k. As we show in the proof of Lemma 3.20, every set $F \in C$ with cardinality k can contribute to at most q^{s-k} positive test graphs in A. This allows us to estimate the number of positive test graphs in A as follows:

$$\begin{aligned}
|A \cap E_+| &\leq \sum_{k=1}^{l} (r-1)^k q^{s-k} \\
&= q^s \sum_{k=1}^{l} \left(\frac{r-1}{q}\right)^k \\
&< q^s \sum_{k=1}^{\infty} \left(\frac{r-1}{q}\right)^k \\
&\leq q^s \sum_{k=1}^{\infty} \left(\frac{1}{3}\right)^k \\
&= q^s/2
\end{aligned}$$

Inserting this estimation together with $|E_+| = q^s$ and the bound on $\delta_+\left(q,r,l\right)$ of Lemma 3.20 into (3.3), we obtain

$$\begin{aligned}
t &\geq \frac{q^s/2}{3q^{s-\lceil (s+1)/2 \rceil}\left(r-1\right)^{\lceil (s+1)/2 \rceil}} \\
&= \frac{1}{6}\left(\frac{q}{(r-1)}\right)^{s/2} \cdot \left(\frac{q}{(r-1)}\right)^{\lceil (s+1)/2 \rceil - s/2} \\
&\geq \frac{1}{6}\left(\frac{q}{(r-1)}\right)^{s/2} \\
&\geq \frac{1}{6}\left(\frac{q}{4s \ln q}\right)^{s/2} \\
&= q^{\Omega(s)}.
\end{aligned}$$

This finishes case 1.

Case 2: $A = \mathbb{B}^{q^2}$. From equation (3.1), we know that for the random test graph G

$$t \geq \frac{\mathbb{P}\left[G \in A\right] - \mathbb{P}\left[G \in A\left(f\right)\right]}{\delta_{-}\left(q, r, l\right)}.$$

When we introduced the random test graph G, we already noted that $\mathbb{P}\left[G \in A\left(f\right)\right] \leq 1/2$ for our choice $\varepsilon = \left(s \ln q + \ln 2\right)/q$. Since in the case $A = \mathbb{B}^{q^2}$ we have $\mathbb{P}\left[G \in A\right] = 1$, the bound of Lemma 3.21 gives us

$$t \geq \frac{1/2}{q^{2l}\left(\varepsilon l\right)^{r}}.$$

For $s \geq 1$ we have $\varepsilon \leq \left(2s \ln q\right)/q$, and then we obtain

$$t \geq \frac{1}{2q^{2s}}\left(\frac{q}{2s^2 \ln q}\right)^{4s \ln q}.$$

Since we assumed $s \leq \frac{1}{12}\sqrt{q/\ln q}$, we have $2s^2 \ln q \leq q/2$, so using $\ln q > (2/3)\log q$ we obtain

$$\begin{aligned} t &\geq \frac{1}{2q^{2s}} 2^{4s \ln q} \\ &> \frac{2^{8s \log q/3}}{2q^{2s}} \\ &= \frac{1}{2} q^{2s/3}. \end{aligned}$$

This finishes case 2 and the proof of the theorem. □

Chapter 4

Branching Programs and Ordered Binary Decision Diagrams

In this chapter we introduce branching programs and ordered binary decision diagrams. While general branching programs are of theoretical interest, the extensive applications of ordered binary decision diagrams have motivated a lot of research in this area. We introduce multilinear circuits as a generalization of ordered binary decision diagrams and nondeterministic read-once branching programs. In the further course of this thesis, we will deal with multilinear circuits as the most general one of these computational models.

4.1 Branching Programs

Branching programs are a model for computation that proceeds over time. Like Boolean circuits, branching programs read input data of fixed size. A branching program assumes a state that controls the computation. Each state transition is determined by the value of an input variable. The computation is terminated when an accepting or rejecting state is reached.

Definition 4.1. A *deterministic branching program* is a directed acyclic graph with one source (a node without predecessor) and two sinks (nodes with fanout 0). All non-sink nodes, also referred to as inner nodes, have fanout two. Every inner node is labeled by a Boolean input variable. The edges leaving a node labelled by the variable x are labeled by x and $\neg x$. The sinks are labelled with 0 or 1 ("rejecting" or "accepting", respectively).

The computation of a branching program starts at the source. At every node, which represents the current state, the variable the node is labelled with is tested. Then the edge which is consistent with the value of the variable is followed. The computation terminates when an accepting or rejecting state is reached.

Deterministic branching programs are an interesting computational model for studying the space required by Turing machines to accept certain languages. In

order to study the space complexity of languages, we turn to the model of *off-line Turing machines*. An off-line Turing machine is a Turing machine with two tapes and two tape heads. The machine operates simultaneously on both tapes. At every transition, it processes the symbols read by both tape heads and decides in which directions to move the tape heads. One of the tapes is designated as the *input tape*, the other tape is the *storage tape*. An off-line Turing machine may only read its input tape and is not allowed to change its contents. The machine may only read the input string and the two neighboring blank symbols, it may not scan areas on the input tape that are not occupied by the input string.

Definition 4.2. For a function $S(n)$, an off-line Turing machine M is $S(n)$ *space-bounded* if M scans at most $S(n)$ cells on its storage tape for all input strings of length n.

The complexity class $DSPACE(S(n))$ consists of all languages that are recognizable by $S(n)$ space-bounded Turing machines. Using off-line Turing machines for defining space complexity enables us to have a sensible notion of sublinear space complexity.

We saw in Section 2.3 that Boolean circuits can simulate Turing machines and that there is a close link between the time complexity of languages and the circuit complexity of the associated functions. It is easy to see that branching programs can also simulate Turing machines in a straightforward way, and it is possible to derive a relationship between the space required by a Turing machine and the number of nodes of the corresponding branching program. The following theorem states the simulation result. A more detailed summary of the relationships between Turing machines and branching programs can be found in [29] or [34].

Theorem 4.3. *If the language $L \subseteq \{0,1\}^*$ is accepted by a $S(n) = \Omega(\log n)$ space-bounded off-line Turing machine, then the series $f_1, f_2, \ldots$ of Boolean functions defined by $f_n^{-1}(1) = L \cap \{0,1\}^n$ is accepted by deterministic branching programs with $2^{O(S(n))}$ nodes.*

Proof. A configuration of an off-line Turing machine consists of the current state, the positions of both of the tape heads and the contents of the storage tape. Since $S(n) = \Omega(\log n)$, the number of cells used on the storage tape is the dominating factor which determines the number of configurations, which leads to $2^{O(S(n))}$ configurations. For every possible configuration of an off-line Turing machine M, we introduce a corresponding inner node of the branching program we construct for M. We label every inner node by the variable which corresponds to the symbol of the input tape which is scanned in the configuration associated with this node. We attach two outgoing edges to every inner node that lead to the configurations the machine assumes depending on the input symbol read. It is easy to see that we do not have to do much work in order to deal with blank symbols in the input. □

As a consequence of Theorem 4.3, a superpolynomial lower bound on the number of nodes of deterministic branching programs for a function in NP would imply $DSPACE(\log n) \neq NP$. However, the best known lower bounds are of the order $\Omega\left(n^2/\log^2 n\right)$. These bounds are still better than any bounds that are known on the combinational complexity of any function in NP. As for Boolean circuits, there has been some success studying restricted versions of the model. A type of restricted branching program that has been heavily studied is the *read-once branching program*.

Definition 4.4. A *read-once branching program* is a branching program in which every input variable occurs at most once on every path from the source to a sink.

We will return to read-once branching programs in the next section. Lower bounds have also been proved for branching programs that can read each variable more than once, but not an unlimited number of times [25, 6, 17].

It is natural to define a nondeterministic variant of branching programs in a similar way as for other computational models. This leads to a more general type of branching program.

Definition 4.5. *Nondeterministic branching programs* are defined like deterministic branching programs, except that their inner nodes are not labelled and have arbitrary fanout, and that edges may be labelled with arbitrary literals. Unlabeled edges are also allowed, and there does not have to be a rejecting sink.

We say a path in a branching program is *consistent* with a certain input if its edges are only labelled by literals that assume the value "true" (unlabeled edges are also allowed). A nondeterministic branching program accepts an input iff there is some path from its source to the accepting sink that is consistent with the input.

It is easy to see that Boolean circuits are at least as efficient as nondeterministic branching programs, i.e. that a function that can be computed by a nondeterministic branching program with few edges can be computed by a Boolean circuit with few gates.

Theorem 4.6. *If a function $f : \mathbb{B}^n \mapsto \mathbb{B}$ can be computed by a nondeterministic branching program with k nodes and l edges, then f can be computed by a Boolean circuit with $l - k$ OR gates and l AND gates.*

Proof (Sketched in [29]). The idea is to associate with every node v of the branching program a gate g_v of the circuit. Then the value computed at the gate g_v should be 1 iff there is a consistent path from the node v to the accepting sink. The gate associated with the source is the output gate of the circuit.

We can build the circuit by introducing gates starting at the accepting sink of the branching program. A topological sort of the branching program can provide a suitable ordering of the nodes. We associate the accepting sink with a gate

yielding the constant 1 and a possible rejecting sink with a gate yielding the constant 0. Let the node v have outgoing unlabeled edges $e_1, e_2, \ldots$ and outgoing edges $e_a, e_b, \ldots$ that are labelled with the literals $a, b, \ldots$, respectively. Let the edges $e_1, e_2, \ldots$ be incident to nodes which are associated with the gates $g_1, g_2, \ldots$ and the edges $e_a, e_b, \ldots$ be incident to nodes which are associated with the gates $g_a, g_b, \ldots$, respectively. Then we introduce the gate g_v which computes

$$g_1 \vee g_2 \vee \ldots \vee a \wedge g_a \vee b \wedge g_b \vee \ldots .$$

If the node v has l_v outgoing edges, then we need $l_v - 1$ OR gates to compute this term. If v only has one outgoing edge which is unlabeled and incident to the node associated with g_1, then we identify g_v with g_1 (g_v and g_1 are then the same gate in the circuit). If g_v only has one outgoing edge that is labelled, then g_v is an AND gate. In the remaining cases g_v is an OR gate. □

4.2 Ordered Binary Decision Diagrams

"*Binary decision diagram (BDD)*" is a different name for "branching program". The term "binary decision diagram" became popular after Bryant [7] had applied ordered BDDs to problems in hardware verification. A variable ordering π on a set of variables $X = \{x_1, \ldots, x_n\}$ is a permutation of the set $\{1, \ldots, n\}$. The list of variables ordered by π is $x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)}$.

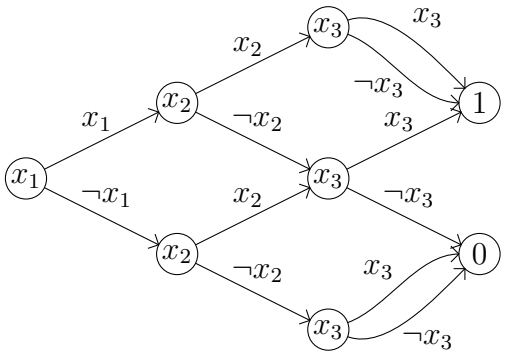
Definition 4.7. An *ordered binary decision diagram (OBDD)* is a deterministic read once branching program in which, on every path from the source to a sink, the variables only occur in the order prescribed by some ordering π that does not depend on the path.

A π-OBDD is an OBDD in which, on every path from the source to a sink, the variables only occur in the order prescribed by the ordering π.

The computational models we defined in the previous sections of this thesis are referred to for proving lower bounds and for suggesting intractability of certain functions. Unlike these, OBDDs are used as a data structure for efficiently representing and manipulating Boolean functions. The following table lists operations and decision problems for Boolean functions that can be efficiently computed when using OBDDs, i.e. in time polynomial in the number of nodes of the OBDD.

Name	Problem	Algorithm
Evaluation	Given a Boolean function $f : \mathbb{B}^n \mapsto \mathbb{B}$ and an argument $\overrightarrow{x} \in \mathbb{B}^n$, compute $f(\overrightarrow{x})$.	There can only be one path from the source to a sink in an OBDD for f that is consistent with $\overrightarrow{x}$. This path can easily be found.
Boolean operation	Given OBDDs for f and g, find an OBDD for the function $f \otimes g$, where $\otimes$ stands for $\vee$, $\wedge$ or some other Boolean operation.	It is possible to work with OBDDs in a similar way as with finite automata, which also allow such Boolean operations.
Complementation	Given an OBDD for f, find an OBDD for $\neg f$.	Exchange the accepting and rejecting sink.
Substitution by constant	Given an OBDD for $f(x_1, \ldots, x_n)$, an index i and a constant $c \in \mathbb{B}$, find an OBDD for the function $f(x_1, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_n)$.	Delete the edges from the OBDD that are labelled with a literal that is false if $x_i = c$. Identify vertices with each other that are connected by edges labelled with a literal that is true if $x_i = c$.
Satisfiability	Given an OBDD for $f : \mathbb{B}^n \mapsto \mathbb{B}$, find some $\overrightarrow{x} \in \mathbb{B}^n$ such that $f(\overrightarrow{x}) = 1$ if such an $\overrightarrow{x}$ exists.	Due to the read-once property, every path from the source to the accepting sink of the OBDD for f is consistent with some input. Thus, it is sufficient to check for paths from the source to the accepting sink.

Several other operations and decision problems can be computed by combining the algorithms sketched in the table. For example, universal or existential quantification of variables as well as equivalence test (given OBDDs for f and g,



This OBDD computes the threshold function $T_2^3(x_1, x_2, x_3)$. The computation proceeds from left to right.

Figure 4.1: An example of an OBDD

decide whether $f = g$) can be performed efficiently. The algorithms available for OBDDs can be more efficient than is apparent from the brief presentation here. Using simple reduction rules, OBDDs can be transformed into a normal form that is unique up to graph isomorphism. This can allow particularly efficient equivalence checks. When performing Boolean operations on OBDDs, isomorphisms of subgraphs can be exploited in order to keep the OBDD representation compact.

The multitude of efficient algorithms available for OBDDs makes OBDDs valuable for a wide range of applications, particularly in hardware synthesis and verification. For example, OBDDs can be used to represent the Boolean function computed by a Boolean circuit that is to be implemented as hardware. Then the OBDD representation of this function can be used for fast evaluation for the purpose of testing and for equivalence tests for verification. OBDDs can also be used to succinctly represent large sets of states that some automaton, implemented as hardware, can assume. Verification then consists in checking if the automaton can reach any forbidden state. For a survey of the applications of OBDDs and branching programs with other restrictions, we refer the reader to the monograph of Wegener [47].

Since OBDDs have to obey severe restrictions on their structure, some important functions that are computable efficiently by Boolean circuits can only be represented inefficiently by OBDDs. The minimum size of an OBDD for a certain function may depend on the variable ordering of the OBDD. The choice of a variable ordering can be a difficult decision that requires sophisticated heuristics. Some functions, such as the threshold function T_k^n, have polynomial size OBDDs for any variable ordering. Other functions, such as the individual result bits obtained by adding two binary coded integers, only have efficient OBDD representations for particular variable orderings.

An example of a function that requires exponential size OBDDs for any variable ordering is the hidden weighted bit function introduced by Bryant [8].

Definition 4.8. The hidden weighted bit function HWB_n is defined on n variables. Let $x_1, \ldots, x_n$ be variables and let $w = |\{x_i \mid 1 \leq i \leq n \text{ and } x_i = 1\}|$. Then the function $HWB_n(x_1, \ldots, x_n)$ assumes the value x_w.

Theorem 4.9 ([8]). *Every OBDD for the function HWB_n has at least $2^{n/6}$ nodes.*

OBDDs are inefficient for many functions of practical importance, such as multiplication. Therefore it is natural to consider branching program models that are less restricted than OBDDs. Deterministic read-once branching programs without any restrictions on variable ordering can be more efficient than OBDDs. Such a branching program is also called *free binary decision diagram (FBDD).*

Theorem 4.10 ([41]). *There are FBDDs for the function HWB_n with less than $3n^2$ nodes.*

However, FBDDs are still inefficient for many functions.

Definition 4.11. The function $MULT_n(x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0)$ is the nth bit from the left of the binary coding of the product of the two integers with the binary codings $x_{n-1} \ldots x_0$ and $y_{n-1} \ldots y_0$.

Theorem 4.12 ([28]). *FBDDs for $MULT_n$ require $2^{\Omega(\sqrt{n})}$ nodes.*

We call a square matrix M with Boolean entries a *permutation matrix* if every row and every column of M contains exactly one entry that is 1.

Definition 4.13. The function $PERM_n$ is defined on n^2 Boolean variables which represent a $n \times n$ Boolean matrix. This function assumes 1 iff this matrix is a permutation matrix.

Theorem 4.14 ([19]). *Every FBDD for the function $PERM_n$ has at least $\Omega(2^n)$ nodes.*

Another possibility to decrease branching program size is to allow nondeterminism. For example, it is easy to find an efficient nondeterministic read-once branching program for the function $\neg PERM_n$ which is 1 iff its argument is not a permutation matrix: nondeterministically choose rowwise or columnwise processing, and then look for a row or column that only contains ones. The resulting branching program has a linear number of nodes. On the other hand, since taking the complement of an FBDD is as simple as for OBDDs, the lower bound of Theorem 4.14 also holds for the function $\neg PERM_n$.

However, there are several lower bounds for nondeterministic read-once branching programs as well. We cite one of them.

Theorem 4.15 ([20]). *Every nondeterministic read-once branching program for the function $PERM_n$ has at least $2^{\Omega(n)}$ nodes.*

In the next chapter we give lower bounds for multilinear circuits, which are a generalization of nondeterministic read-once branching programs. In the search of data structures that simultaneously offer efficient algorithms and succinct representations of important functions, many other branching program models have been proposed. For more information on these models we again refer the reader to the monograph of Wegener [47].

4.3 Multilinear Circuits

The read-once property is a fundamental constraint that makes ordered binary decision diagrams interesting. Recall that a branching program can be regarded as a restricted kind of Boolean circuit (Theorem 4.6). A restriction on Boolean circuits that corresponds to the read-once property for branching programs has been introduced as *multilinearity*. Circuits that comply with this restriction are called *multilinear circuits*. This circuit model is a generalization of nondeterministic read-once branching programs.

Since the term "multilinear" has been first used to describe a restriction on arithmetic circuits, we discuss arithmetic multilinear circuits before turning to Boolean multilinear circuits. An *arithmetic circuit* performs computations in a field. The gates of the circuit compute the field operations $+$ and $\times$. The inputs of the circuit are variables and field elements. A polynomial is multilinear if in each of its monomials the power of every variable is at most one. An arithmetic circuit is multilinear if every polynomial computed by some gate of the circuit is multilinear. Multilinear arithmetic circuits were defined in [24]. Raz [31] proved a superpolynomial gap between the size of multilinear arithmetic circuits and the size of multilinear arithmetic formulas.

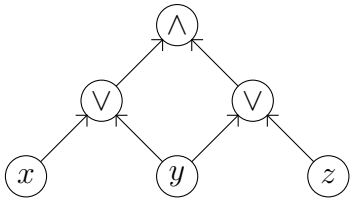
Raz [30] introduced syntactic multilinear circuits which are slightly more restricted than multilinear circuits. In order to define syntactic multilinear circuits, let $var(g)$ be the set of variables that occur in the subcircuit rooted at the gate g of some circuit. An arithmetic circuit is syntactic multilinear if $var(g_1) \cap var(g_2) = \emptyset$ for each of its $\times$-gates with inputs g_1 and g_2. Every syntactic multilinear circuit is multilinear, but not vice versa. Raz [30] showed that multilinear *formulas* can be converted to syntactic multilinear formulas without an increase in size.

We now turn to Boolean multilinear circuits. In order to define Boolean multilinear circuits, let $var(g)$ again be the set of variables that occur in the subcircuit rooted at the gate g of some circuit.

Definition 4.16. A Boolean circuit is multilinear if $var(g_1) \cap var(g_2) = \emptyset$ for each of its AND gates g with inputs g_1 and g_2.

In other words, a Boolean circuit is multilinear if the inputs to each of its AND gates are computed from disjoint sets of variables. Our definition of Boolean

multilinear circuits is equivalent to the definition of multilinear circuits in [40]. This notion of Boolean multilinear circuits closely mimics the definition of arithmetic syntactic multilinear circuits. In [27] a slightly less restrictive definition of Boolean multilinear circuits is used which resembles the concept of arithmetic multilinear circuits more closely. While it may be possible to prove our results about multilinear circuits using the more general definition of [27], this appears to require significantly more sophisticated proofs, so we decide to limit ourselves to the more restrictive notion of multilinearity.



According to Definition 4.16, this circuit is not multilinear because the variable y is an input to both OR gates, and hence y occurs in both subcircuits rooted at the inputs to the AND gate.

Figure 4.2: An example of a circuit that is not multilinear

It is clear that every Boolean function f can be computed by a multilinear circuit with $|PI(f)|-1$ OR gates: just take the DNF of f. It is easy to see that the transformation of Theorem 4.6 applied to a read-once branching program yields a multilinear circuit. So every function with an efficient nondeterministic read-once branching program also has a just as succinct multilinear circuit. Thus, many functions commonly referred to have multilinear circuits that are much smaller than their DNFs. Consider the threshold function T_k^n as an example. The threshold function T_k^n has $\binom{n}{k}$ prime implicants, but can be computed by a multilinear circuit of size $O(nk)$. The construction of an efficient ordered binary decision diagram for T_k^n can be found in [47, chapter 4]. Hence, the gap between the size of a smallest multilinear circuit which computes a certain function and the size of the DNF of this function can be exponential.

Let $CONN(n)$ be the function whose argument is the adjacency matrix of a directed n-vertex graph and assumes the value 1 if and only if the graph is connected. Sengupta and Venkateswaran have proved the following theorem.

Theorem 4.17 ([40]). *Multilinear circuits for $CONN(n)$ have at least $\sqrt{\frac{1}{n} \cdot \left(\frac{4}{3}\right)^{n-1}}$ gates.*

Since the function $CONN(n)$ can be computed by monotone circuits of polynomial size, this shows that the gap between multilinear complexity and monotone complexity is also exponential. Let $BPM(n)$ be the function which decides if an n-vertex bipartite graph has a perfect matching. The following theorem is from Ponnuswami and Venkateswaran.

Theorem 4.18 ([27]). *Multilinear circuits for $BPM(n)$ have at least $\Omega(2^{0.459n})$ gates.*

In the next chapter we prove lower bounds for multilinear circuits of some other functions. While we use a slightly less general notion of multilinearity than in [27], we are able to prove stronger lower bounds which are even optimal.

The following lemma allows us to restrict ourselves to *monotone* multilinear circuits. It is a special case of a theorem given in [12] for read-once nondeterministic machines. We give an alternative proof that uses the specific restrictions of multilinear circuits.

Lemma 4.19. *If f is a monotone function, then any optimal multilinear circuit for f is monotone.*

Proof. Let S be an optimal multilinear circuit for f. We take the notion of a parse-graph G of S from [40]: The parse-graph G includes the output of S; for any OR gate v of G, exactly one immediate predecessor of v is included as its only predecessor in G; and for any AND gate v included in G, both immediate predecessors are included as predecessors of v in G. The parse-graph G can be viewed as a kind of circuit that accepts a subset of the inputs that S accepts. Since S is multilinear, a variable can occur at most once in G, so a variable and its negation can never both appear in G. This means that the conjunction of all variables and negated variables in G is consistent, and an implicant of f. So the set of all non-negated variables in G must contain a prime implicant of f.

Every input that a circuit accepts is accepted by one of its parse-graphs. Therefore, we can set all inputs of a multilinear circuit for f that are fed from negated variables to 1. Clearly, the variable set of every parse-graph of the resulting circuit will still contain a prime implicant of f because f is monotone. □

Chapter 5

Multilinear Circuits Are Inefficient for Union-Free Functions

In this chapter we introduce union-free functions. The clique function and the polynomial function, which we have already studied thoroughly in Chapter 3, are examples of union-free functions. We prove an optimal lower bound for multilinear circuits of union-free functions. This bound states that a multilinear circuit for a union-free function needs just as many OR gates as the DNF of the function.

5.1 Union-Free Functions

The following definition of union-free functions will allow us to prove optimal lower bounds for multilinear circuits.

Definition 5.1. A monotone Boolean function is *union-free* if the union of any two of its prime implicants does not contain a new prime implicant.

The clique function is a prominent example of a union-free function.

Lemma 5.2. *The function $CLIQUE(n, s)$ is union-free.*

Proof. Suppose the union of two distinct s-cliques A and B contains all edges of some third clique C. Since all three cliques are distinct and have the same number of vertices, C must contain a vertex u which does not belong to A and a vertex v which does not belong to B. This already leads to a contradiction because either the vertex u (if $u = v$) or the edge $\{u, v\}$ (if $u \neq v$) of C would remain uncovered by the cliques A and B. □

For certain parameters the polynomial function is another example of a union-free function.

Lemma 5.3. *If $s \leq q/2$, then the function $POLY(q, s)$ is union-free.*

Proof. The prime implicants of the function $POLY(q,s)$ are of the form $\bigwedge_{i \in GF(q)} x_{i,f(i)}$ for some polynomial $f(z)$ of degree at most $s-1$. The function $POLY(q,s)$ is *s-disjoint*, i.e. two distinct prime implicants of this function cannot have s variables in common. Otherwise, two distinct polynomials of degree at most $s-1$ would assume the same values at s points, which is impossible.

To prove the lemma, assume that p_1, p_2 and p_3 are distinct prime implicants of $POLY(q,s)$ and that $p_3 \subseteq p_1 \cup p_2$. Then p_3 must have $q/2 \geq s$ variables in common with p_1 or p_2, a contradiction. □

5.2 The Lower Bound for Multilinear Circuits

We now give the lower bound for multilinear circuits:

Theorem 5.4. *Let f be a monotone union-free function. Then any multilinear circuit for f must have at least $|PI(f)|-1$ OR gates.*

It is well known that the minimal DNF of a monotone function is the disjunction of all of its prime implicants.

Corollary 5.5. *Multilinear circuits for $CLIQUE(n,s)$ require $\binom{n}{s}-1$ OR gates (just as many as the DNF of this function).*

Because nondeterministic read-once branching programs can be simulated by multilinear circuits, the bound of $\exp(\Omega(s\log(n/s)))$ given by Corollary 5.5 improves the bound of $\exp(\Omega(\min(s,n-s)))$ given in [6] for nondeterministic read-once branching programs computing $CLIQUE(n,s)$.

Corollary 5.6. *If $s \leq q/2$, then any multilinear circuit for $POLY(q,s)$ has q^s-1 OR gates (just as many as the DNF of this function).*

For the proof of Theorem 5.4 we first give a lemma that describes a restriction of multilinear circuits. This restriction leads to exponential lower bounds for certain monotone Boolean functions. Given a prime implicant p, we show that, depending on the circuit, certain variables of p can be substituted by some variables of another prime implicant p'. This yields a "derived" implicant of the function computed by the circuit. If the function is union-free, we are able to reason further about the derived implicant.

We say a path from a gate to the output of a circuit is *consistent* with a monom m if m is an implicant of all the functions computed at the gates along this path. We call a gate g *necessary* for an implicant m of a circuit S if m is not an implicant of the circuit $S_{g\to 0}$ we obtain from S by replacing g with the constant 0.

Lemma 5.7. *For every gate g which is necessary for an implicant m of S, there is a path from the output of S to g which is consistent with m.*

Proof. First note that m is an implicant of the function computed by an OR gate h iff m is an implicant of one of the inputs to h. Analogously, the implicant m is an implicant of an AND gate h iff m is an implicant of both of the inputs to h.

We find a consistent path in S from the output to g by descending into the circuit starting at the output. Doing so, we compare the two circuits S and $S_{g\to 0}$ with each other. We require that our path consists of gates that are not implied by m in the modified circuit $S_{g\to 0}$. We start with the output gate as the first gate of the path. Assume we have followed the path $g_1, \ldots, g_i$ and we are not done since $g_i \neq g$. We must pick the next gate g_{i+1} on our path. If g_i is an OR gate, then we choose g_{i+1} as the input to g_i that is implied by m in S. Since both inputs to g_i are not implied by m in $S_{g\to 0}$, our choice of g_{i+1} is also not implied by m in $S_{g\to 0}$, as we require. If g_i is an AND gate, then we choose g_{i+1} as the input to g_i that is not implied by m in $S_{g\to 0}$. Since both inputs to g_i are implied by m in S, our choice of g_{i+1} is also implied by m in S, as we require.

Finally we must reach g while constructing the path, since every leaf node in $S_{g\to 0}$ which is not g does not differ from the corresponding node in S. □

Let $PI_g(f)$ denote the set of prime implicants of f that g is necessary for. By $PI(g)$ we denote the set of prime implicants of the function computed at gate g.

Lemma 5.8 (Exchange Lemma). *Let g be a gate in a monotone multilinear circuit S for a function f and p, p' be prime implicants in $PI_g(f)$. Let $m \subseteq p$ and $m' \subseteq p'$ be distinct prime implicants in $PI(g)$.*

(i) If w is a path from g to the output of S that is consistent with p, then w is consistent with the derived monom $(p \setminus m) \cup m'$. This means in particular that the derived monom $(p \setminus m) \cup m'$ is also an implicant of f.

(ii) If f is union-free, then the identity $p = (p' \setminus m') \cup m$ holds.

Proof. (i) We first note that the substitution of the variables of m by the variables of m' is valid at gate g. Then we observe that the substitution remains valid along the path w due to the multilinearity of the circuit.

We have to show that $(p \setminus m) \cup m'$ is an implicant of all functions computed along w $(g = g_1, \ldots, g_t)$. We prove this by induction on the length of the path w. For $g_1 = g$ the claim is correct since $(p \setminus m) \cup m'$ is a superset of $m' \in PI(g_1)$. For the inductive step, assume that $q \in PI(g_i)$ such that $q \subseteq (p \setminus m) \cup m'$. If g_{i+1} is an OR gate, then q is an implicant of g_{i+1}. If g_{i+1} is an AND gate, then let h be the other gate feeding it. We know that p is an implicant of the function computed at g_{i+1}. Hence, there must be some $m_h \in PI(h)$ such that $m_h \subseteq p$. Because the circuit is multilinear, we have $var(g_i) \cap var(h) = \emptyset$. Gate g belongs to the subcircuit rooted at gate g_i. We conclude that $var(g) \subseteq var(g_i)$ and that $var(g) \cap var(h) = \emptyset$. Since a variable of a prime implicant of a gate must occur somewhere in the subcircuit rooted at that gate, we conclude from $m \in PI(g)$ and $m_h \in PI(h)$ that $m \cap m_h = \emptyset$. Now we can see that $q \cup m_h$, an implicant of the function computed at g_{i+1}, is a subset of $(p \setminus m) \cup m'$.

(ii) According to Lemma 5.7, there is path from g to the output of S that is consistent with p, because g is necessary for p. Therefore, according to (i), the monom $(p \setminus m) \cup m'$ is an implicant of f. Clearly, we have $(p \setminus m) \cup m' \subseteq p \cup p'$. Since f is union-free, this implies $p \subseteq (p \setminus m) \cup m'$ or $p' \subseteq (p \setminus m) \cup m'$. Because m and m' are distinct prime implicants, we have $m \not\subseteq m'$ and $m \not\supseteq m'$. The inclusion $p \subseteq (p \setminus m) \cup m'$ is impossible because $m \not\subseteq m'$. So $p' \subseteq (p \setminus m) \cup m'$ holds, this implies $m' \supseteq p' \setminus p$.

Since its assumptions are symmetrical, claim (i) also implies that $(p' \setminus m') \cup m$ is an implicant of f. Arguing in the same way as above we conclude that $p \subseteq (p' \setminus m') \cup m$. Since $m' \supseteq p' \setminus p$, we have $(p' \setminus m') \cup m \subseteq p$. From the two inclusions $p \subseteq (p' \setminus m') \cup m$ and $(p' \setminus m') \cup m \subseteq p$ we derive $p = (p' \setminus m') \cup m$. □

We now show how to transform a multilinear circuit for a union-free function into a normal form. We call a monotone circuit *broom-like* if, for each of its AND gates with inputs g_1 and g_2, $|PI(g_1)| = 1$ or $|PI(g_2)| = 1$ (or both). Thus, broom-like circuits have a particularly simple structure, and there is a direct correspondence between their prime implicants and their OR gates.

Lemma 5.9. *Every monotone multilinear circuit S for a union-free function f can be transformed into a broom-like formula for f with at most as many OR gates as S.*

Proof. We first transform S into a broom-like multilinear *circuit* for f without an increase in the number of OR gates. For this we need to know the following.

Claim 5.10. *Let g be an AND gate with inputs g_1 and g_2. If $PI(g_1)$ is not empty, then there exists a monom m in $PI(g_1) \cup PI(g_2)$ such that $m \subseteq p$ for all $p \in PI_g(f)$.*

Proof. Suppose there is no suitable m in $PI(g_1)$. We show that then there must be an m in $PI(g_2)$ such that $m \subseteq p$ for all p in $PI_g(f)$. Since there is no suitable m in $PI(g_1)$, $PI_g(f)$ cannot be empty. We pick some arbitrary p' in $PI_g(f)$. Because p' is an implicant of the function computed at g, there must be some m_2' in $PI(g_2)$ such that $m_2' \subseteq p'$. We prove that in fact

$$m_2' \subseteq p \text{ for all } p \in PI_g(f) \ .$$

We distinguish two cases. First note that there must be some m_1' in $PI(g_1)$ such that $m_1' \subseteq p'$.

Case 1: $m_1' \not\subseteq p$. Then there is some m_1 in $PI(g_1)$ such that $m_1 \subseteq p$, since p is an implicant of the function computed at g. Since g is an AND gate, the input g_1 is also necessary for p. Therefore we can apply Lemma 5.8(ii), which yields that $p = (p' \setminus m_1') \cup m_1$. Hence, $m_2' \subseteq p$ because $m_2' \subseteq p'$ and $m_1' \cap m_2' = \emptyset$ due to the multilinearity of the circuit.

Case 2: $m_1' \subseteq p$. Note that there must be some $p'' \in PI_g(f)$ such that $m_1' \not\subseteq p''$ because, by our initial assumption, $m_1' \in PI(g_1)$ cannot be a suitable

choice of m. Case 1 applies to p'' because $m_1' \not\subseteq p''$, and we conclude $m_2' \subseteq p''$. There must be some m_1'' in $PI(g_1)$ with $m_1'' \subseteq p''$. We use Lemma 5.8 again and find that $p = (p'' \setminus m_1'') \cup m_1'$. Hence, $m_2' \subseteq p$ because $m_2' \subseteq p''$ and $m_1'' \cap m_2' = \emptyset$ due to the multilinearity of the circuit. □

We describe a modification that can be applied to every AND gate g which prevents S from being broom-like. Let g_1 and g_2 be the gates that feed g. The gate g prevents S from being broom-like, so $|PI(g_1)| > 1$ and $|PI(g_2)| > 1$. Let m be the monom in $PI(g_i)$ ($i \in \{1,2\}$) given by Claim 5.10. We add a new gate h that computes m (along with the corresponding subcircuit for this computation). Then we disconnect g from g_i and feed g from h instead of g_i. Clearly, the resulting circuit S' rejects all the inputs that the original circuit rejected, since we are dealing with monotone circuits. Because S' accepts all inputs that $S_{g\to 0}$ accepts, g must be necessary for any prime implicant p of S that is not a prime implicant of S'. But according to Claim 5.10, after the modification every such p remains an implicant of the function computed at g. This way we obtain a broom-like multilinear circuit S^* for f without an increase in the number of OR gates.

We now describe a way of transforming a broom-like multilinear circuit S^* for f into a broom-like formula F for f without an increase in the number of OR gates.

Claim 5.11. *Let g be a gate in S^* such that $PI(g) \neq \emptyset$. Then*

(i) there is some monom m in $PI(g)$ such that $m \subseteq p$ for all p in $PI_g(f)$, or

(ii) there is some path w from g to the output of S^* that is consistent with all $p \in PI_g(f)$.

Proof. We show that if (i) does not hold, then (ii) follows. This proof has a similar structure compared to the proof of the Claim 5.10. Since (i) does not hold, $PI_g(f)$ cannot be empty. So there is some $p' \in PI_g(f)$ and, according to Lemma 5.7, some path w' from g to the output of S^* that is consistent with p'. We prove that in fact

$$w' \text{ is consistent with } p \text{ for all } p \in PI_g(f) \ .$$

We distinguish two cases. First note that there is some $m' \in PI(g)$ with $m' \subseteq p'$ because p' is an implicant of the function computed at g.

Case 1: $m' \not\subseteq p$. There must be some $m \in PI(g)$ such that $m \subseteq p$ because p is an implicant of the function computed at g. Lemma 5.8 yields that $p = (p' \setminus m') \cup m$ and that w' is consistent with p.

Case 2: $m' \subseteq p$. Because (i) does not hold, there is some p'' in $PI_g(f)$ such that $m' \not\subseteq p''$. Case 1 applies to p'' because $m' \not\subseteq p''$, and we conclude that w' is consistent with p''. There must be some m'' in $PI(g)$ with $m'' \subseteq p''$. Lemma 5.8 tells us that $p = (p'' \setminus m'') \cup m'$ and that w' is consistent with p. □

We now describe a modification that we carry out for every gate g of S^* with fanout larger than 1 in order to reduce its fanout to 1. As with the modification for making the circuit broom-like, we only have to check the prime implicants for which g is necessary. The pathological case $PI(g) = \emptyset$ ($g = 0$) is trivial, so it suffices to discuss the two cases listed in Claim 5.11.

Case 1: There is some m in $PI(g)$ such that $m \subseteq p$ for all p in $PI_g(f)$. We remove g from the circuit and replace all wires from g by subcircuits that each compute m. The resulting circuit computes a function that is clearly implied by all prime implicants p in $PI_g(f)$.

Case 2: There is some path w from g to the output of S^* that is consistent with all p in $PI_g(f)$. We then cut all wires stemming from g that are not on path w, i.e. we replace inputs to other gates from g by the constant 0. All prime implicants in $PI_g(f)$ are preserved because after the modification w is still consistent with all of them. To see this, note that, due to the multilinearity of the circuit, every AND gate on w can have at most one input that depends on g (such an input must be on w itself). □

The following lemma enables us to count the prime implicants of monotone functions by counting the OR gates of their monotone broom-like formulas.

Lemma 5.12. *Let F be a monotone broom-like formula computing f. Then F has at least $|PI(f)| - 1$ OR gates.*

Proof. We prove the lemma by induction on the size of the formula. If F does not contain any OR gates, it is clear that the claim holds. Let F_1 and F_2 be formulas computing the monotone functions f_1 and f_2, respectively. Since

$$|PI(f_1 \vee f_2)| \leq |PI(f_1)| + |PI(f_2)| ,$$

$$|PI(f_1 \vee f_2)| - 1 \leq ((|PI(f_1)| - 1) + (|PI(f_2)| - 1)) + 1 ,$$

so the claim holds for $F_1 \vee F_2$. So let us turn to the case of conjunction. W.l.o.g. let f_1 be a monom. Then

$$|PI(f_1 \wedge f_2)| \leq |PI(f_2)| ,$$

so the claim holds in this case too. □

Theorem 5.4 follows immediately from Lemma 5.9 together with Lemma 5.12. Recall that, according to Lemma 4.19, it is enough to consider monotone multilinear circuits.

Chapter 6

An Upper Bound for the Clique Function

In this chapter we show that the union-freeness property is not sufficient for proving good lower bounds for unrestricted monotone circuits. By Corollary 5.5, the function $CLIQUE(n, n-1)$ requires $n-1$ OR gates to be computed by a multilinear circuit. On the other hand, we prove the following upper bound in this chapter.

Theorem 6.1. *The function $CLIQUE(n, n-1)$ can be computed by a monotone formula with $O(\log n)$ OR gates.*

Thus, general monotone circuits for the clique function can be much more efficient than multilinear circuits. The only other upper bound for the clique function that we are aware of is given in [46] and is only for its non-monotone complexity.

We will use error correcting codes for constructing a circuit for the clique function that is more efficient than the DNF. In order to introduce codes, we first need to define the Hamming distance between two words.

Definition 6.2. Let A be a set of symbols. For two words $x, y \in A^k$, $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$, the *Hamming distance* between x and y is

$$d(x, y) = |\{i \mid x_i \neq y_i \text{ and } 1 \leq i \leq k\}| .$$

Definition 6.3. A code over an alphabet A with *block length* k and *minimal distance* d is a nonempty subset C of A^k such that $d = \min\{d(x, y) \mid x, y \in C\}$.

For our upper bound on monotone complexity, we will need a code with sufficient minimal distance and number of code words. To prove its existence, we will use the Gilbert bound for codes. An introduction to the theory of coding, including the Gilbert bound, can be found in the textbook of van Lint [44].

Theorem 6.4 (Gilbert Bound). *For every alphabet A, every $k \in \mathbb{N}$ and every $d \in \mathbb{N}$ such that $1 \leq d \leq k$, there exists a code C over A with block length k, minimal distance d and*

$$|C| \geq |A|^k \Big/ \sum_{i=0}^{d-1} \binom{k}{i} (|A|-1)^i$$

codewords.

Proof. Consider a code $C \subseteq A^k$ with minimal distance at least d such that no codeword can be added without making its minimal distance smaller than d. For every word $w \in A^k$ there must be a codeword $c \in C$ such that $d(w,c) < d$. Otherwise some word w could be added to the code without making the minimal distance smaller than d. So A^k is the union of all words with hamming distance at most $d-1$ to some codeword in C. For every word $c \in A^k$ we have

$$\left|\left\{w \in A^k \mid d(w,c) \leq d-1\right\}\right| = \sum_{i=0}^{d-1} \binom{k}{i} (|A|-1)^i .$$

This gives us

$$|A|^k \leq |C| \sum_{i=0}^{d-1} \binom{k}{i} (|A|-1)^i ,$$

and the bound of the theorem follows. □

Lemma 6.5. *For every k, there is a code $C \subseteq A^k$ over an alphabet A ($|A| = 2^{12}$) with block length k, minimal distance $d > 3k/4$ and at least $|C| \geq 2^k$ codewords.*

Proof. We choose some alphabet A with 2^{12} symbols. According to the Gilbert bound, there is a code C with minimal distance d and

$$|C| \geq \frac{2^{12k}}{d2^k \cdot 2^{12(d-1)}} = 2^{12(k-d+1)-k-\log d}$$

codewords. We choose $d = 3k/4 + 1$ and obtain $|C| \geq 2^{3k-k-\log d}$ and $|C| \geq 2^k$ since $d \leq k$. □ □

We will use the following lemma for proving the upper bound.

Lemma 6.6. *Let G be a graph with n vertices. If its complement $\overline{G}$ does not contain a triangle and does not have two edges which are not incident to a common vertex, then G has an $(n-1)$-clique.*

Proof. Suppose G does not have an $n-1$-clique. Then $\overline{G}$ is not a star. Suppose $\overline{G}$ does not have two edges which are not incident to a common vertex. Choose arbitrary distinct edges e_1 and e_2 in $\overline{G}$. Let e_1 and e_2 be incident to the common

vertex u. Since $\overline{G}$ is not a star, there is an edge e_3 which is not incident to u. Let e_2 and e_3 be incident to the common vertex $v \neq u$. The edges e_1 and e_3 must share the common vertex w, which is distinct from u and v. Hence, u, v and w form a triangle in $\overline{G}$. □

We are now ready to prove Theorem 6.1.

Proof of Theorem 6.1. To design the desired formula for $CLIQUE\,(n, n-1)$ we use an error correcting code $C \subseteq A^k$ for some k over an alphabet A with a constant number of symbols (independent of n) such that $|C| \geq n$ and the minimal distance d of C is larger than $3k/4$. The existence of such a code of length $k = O(\log n)$ is guaranteed by Lemma 6.5.

We assign to each vertex x (and hence, to each $(n-1)$-clique $V \setminus \{x\}$) its own codeword $code\,(x) \in C$. For each $1 \leq i \leq k$ and $a \in A$, let $S_{i,a}$ be the intersection of all $(n-1)$-cliques whose codes have symbol a in the i-th position. Hence,

$$S_{i,a} = V \setminus \{x \in V \mid code\,(x) \text{ has symbol } a \text{ in position } i\}\ . \tag{6.1}$$

Let $m_{i,a}$ be the monom consisting of all variables which correspond to edges having both their endpoints in $S_{i,a}$ (if $|S_{i,a}| \leq 1$, we set $m_{i,a} = 1$). We claim that the formula

$$F = \bigwedge_{i=1}^{k} \bigvee_{a \in A} m_{i,a}$$

computes $CLIQUE\,(n, n-1)$. Clearly, this formula has $k\,(|A| - 1) = O(\log n)$ OR gates. Using distributivity we obtain the following representation of the function computed by F:

$$F = \bigvee_{(a_1,\ldots,a_k) \in A^k} \bigwedge_{i=1}^{k} m_{i,a_i}\,. \tag{6.2}$$

Every $(n-1)$-clique $V \setminus \{x\}$ with $code\,(x) = (a_1, \ldots, a_k)$ is accepted by the monom $\bigwedge_{i=1}^{k} m_{i,a_i}$ because the clique $V \setminus \{x\}$ contains all the cliques S_{i,a_i}, $i = 1, \ldots, k$. Hence, by (6.2) every $(n-1)$-clique is accepted by F. It remains to show that F does not accept any graph without an $(n-1)$-clique. Let G be a graph accepted by F. Then by (6.2) there is a sequence $a_1, \ldots, a_k$ of symbols in A such that G is accepted by the monom $\bigwedge_{i=1}^{k} m_{i,a_i}$. For a vertex $x \in V$, let

$$P_x = \{i \mid code\,(x) \text{ has symbol } a_i \text{ in position } i\}\ .$$

Since the code C has minimal distance $d > 3k/4$, this implies that for every two distinct vertices x and y,

$$|P_x \cap P_y| \leq k - d < k/4\,. \tag{6.3}$$

Let $\{x, y\}$ be an edge of the complement graph $\overline{G}$. Then the edge $\{x, y\}$ cannot belong to any of the monoms $m_{1,a_1}, \ldots, m_{k,a_k}$, implying that $x \notin S_{i,a_i}$ or $y \notin S_{i,a_i}$ for all $i = 1, \ldots, k$. According to (6.1) this means that for all $i = 1, \ldots, k$, $code\,(x)$ or $code\,(y)$ has symbol a_i at position i. So we have

$$P_x \cup P_y = [k] = \{1, \ldots, k\} \; . \tag{6.4}$$

Now we are able to show that G must contain an $(n-1)$-clique. We do so by showing that its complement $\overline{G}$ does not contain a triangle and does not contain a pair of vertex disjoint edges. The result then follows with Lemma 6.6.

Assume first that $\overline{G}$ contains a triangle with vertices u, v and w. By (6.4), we have that $P_u \cup P_w = [k]$ and $P_v \cup P_w = [k]$. Taking the intersection of these two equations yields

$$(P_u \cap P_v) \cup P_w = [k] \, .$$

But by (6.3), we have that $|P_u \cap P_v| < k/4$, so $|P_w| > 3k/4$. Similarly we obtain $|P_u| > 3k/4$, implying that $|P_u \cap P_w| > k/2$, a contradiction with (6.3).

Assume now that $\overline{G}$ contains a pair of vertex disjoint edges $\{u, v\}$ and $\{x, y\}$. By (6.4), we have $P_u \cup P_v = [k]$ and $P_x \cup P_y = [k]$. Assume w.l.o.g. that $|P_u| \geq |P_v|$. Then $|P_u| \geq k/2$. We know that

$$P_u = P_u \cap [k] = P_u \cap (P_x \cup P_y) = (P_u \cap P_x) \cup (P_u \cap P_y) \, .$$

Assume w.l.o.g. that $|P_u \cap P_x| \geq |P_u \cap P_y|$. Then $|P_u \cap P_x| \geq |P_u|/2 \geq k/4$, a contradiction with (6.3). □

Theorem 6.1 tells us that monotone circuits for the function $CLIQUE\,(n, n-1)$ are compressible, i.e. for sufficiently large n they require less OR gates than the respective DNF. We now show that monotone circuits for most other clique functions are also compressible. Thus, our optimal lower bounds for the clique function from the previous section cannot be extended to general monotone circuits in any way.

Corollary 6.7. *There exists some s_0 such that, for all $s \geq s_0$ and $n > s$, the function $CLIQUE\,(n, s)$ has monotone circuits with less OR gates than the DNF of this function.*

Proof. We pick s_0 such that every function $CLIQUE\,(s+1, s)$ with $s \geq s_0$ is compressible. This is possible according to Theorem 6.1. Now we show that a clique function $CLIQUE\,(n, s)$ with $s \geq s_0$ and $n > s$ is compressible. We choose arbitrary $s+1$ vertices in the n-vertex graph taken by $CLIQUE\,(n, s)$. Let $p_1, \ldots, p_{s+1}$ be the prime implicants that correspond to the s-cliques of the $s+1$ chosen vertices. We denote the other prime implicants of $CLIQUE\,(n, s)$ by $q_1, \ldots, q_k$, so we can write the DNF of $CLIQUE\,(n, s)$ as

$$CLIQUE\,(n, s) = \left(\bigvee_{i=1}^{s+1} p_i \right) \vee \left(\bigvee_{i=1}^{k} q_i \right) \; . \tag{6.5}$$

Here the disjunction $\bigvee_{i=1}^{s+1} p_i$ is the DNF of the function $CLIQUE\,(s+1, s)$, so this term can be computed by a monotone circuit with less than s OR gates. Hence, according to (6.5) the function $CLIQUE\,(n, s)$ can be computed by a monotone circuit with less OR gates than the DNF of this function. $\square$

Chapter 7

Lower Bounds for Monotone Σ_4-Circuits

In this chapter we study circuits of bounded depth. A circuit has *alternation depth* d iff d is the highest number of blocks of OR gates and blocks of AND gates on paths from input gates to the output. When determining depth, we do not pay attention to NOT gates.

Definition 7.1. A Σ_d-circuit (respectively, Π_d-circuit) is a Boolean circuit with alternation depth at most d such that the output gate is an OR gate (AND gate, respectively).

Good lower bounds for non-monotone bounded depth circuits have been proved [14, 15]. In this chapter we only deal with monotone bounded depth circuits. We contrast the upper bound for the clique function proved in the previous chapter with a lower bound for functions that are even harder than the clique function in a certain sense. We introduced the polynomial function $POLY\,(q, s)$ in Section 3.1. For some polynomial functions we give incompressibility results, similar to those for multilinear circuits, also for monotone Σ_4-circuits. We show that monotone Σ_4-circuits for these functions require at least as many OR gates as the respective DNFs. The construction used in the proof of Theorem 6.1 yields a monotone Π_3-formula. A monotone Π_3-formula is a simple kind of monotone Σ_4-circuit. Thus, to prove upper bounds for the functions we study in this chapter, we would have to give a more elaborate construction than we did for the clique function. A monotone circuit for any of these functions that is more efficient than the DNF would have to be more complicated than a Σ_4-circuit. Therefore, these hard polynomial functions we investigate here are an interesting starting point for looking for new lower bounds. It is not even clear whether these polynomial functions can be computed by unrestricted circuits that are smaller than the respective DNFs.

We will use the following property for proving the lower bound.

Definition 7.2. A Boolean function is *s-disjoint* if any two of its prime implicants do not have s variables in common.

The following lemma shows that the union-freeness property is a special case of the disjointness property.

Lemma 7.3. *Let $p_1, ..., p_r$ be prime implicants of a monotone Boolean function f and m be an implicant of f. Let f be k-homogeneous and k/r-disjoint.*

(i) If $\bigcup_{i=1}^{r} p_i \supseteq m$, then $m \supseteq p_i$ for some i.

(ii) If $x_1, \ldots, x_r$ are variables such that $x_i \in p_i$ and $x_i \notin p_j$ for $i \neq j$, then $\bigcup_{i=1}^{r} (p_i \setminus \{x_i\})$ is not an implicant of f.

Proof. (i) There must be some prime implicant p of f with $m \supseteq p$. Since $\bigcup_{i=1}^{r} p_i \supseteq p$, p must share at least k/r variables with some p_i. Because f is k/r-disjoint, this implies $p = p_i$. Claim (ii) is a direct consequence of (i). □

When stating the following lemma, we deviate from the circuit model introduced in Section 2.2 by using gates of *unbounded* fanin. The lemma deals with Π_3-circuits with gates of unbounded fanin. We restrict these circuits to depth 3. We require the output gate to be an AND gate (possibly with only one input) and the inputs to this gate to be OR gates. The *top fanin* is the fanin of the output gate. The *bottom fanin* is the maximal fanin of the AND gates representing Π_1-subcircuits (if there are no such subcircuits, we define the bottom fanin to be 1).

Lemma 7.4. *Let f be a monotone k-homogeneous and s-disjoint function. If $r \leq k/2s$ and h is a function such that $h \leq f$ (i.e., f evaluates to 1 if h does) and $|PI(h) \cap PI(f)| \geq r$, then any monotone Π_3-circuit for h with bottom fanin at most $s-1$ must have top fanin at least $(k/2s)^r$.*

Proof. Let S be a monotone Π_3-circuit with top fanin a and bottom fanin at most $s-1$, and let F be the function computed by S. Let $a < (k/2s)^r$. We now show that the circuit S must then make an error, i.e. that $F \neq h$. For the sake of contradiction, assume that $F = h$.

We choose arbitrary distinct prime implicants $p_1, \ldots, p_r \in PI(h) \cap PI(f)$. Our goal is to pick $x_1 \in p_1, \ldots, x_r \in p_r$ suitable for Lemma 7.3(ii). Lemma 7.3(ii) then yields a monom m which, according to the Lemma, is not an implicant of h, but for which we show that it is an implicant of F. This way we obtain $F \neq h$ and contradict our assumption.

We pick the x_i's in the order indicated by their indices. During this process we consider the preliminary monoms

$$m_t = \bigcup_{i=1}^{t} (p_i \setminus \{x_i\}) \ , \ t = 1, \ldots, r \, .$$

The preliminary monom m_t is available after the t-th step of the process. Finally, the monom $m = m_r$ is the desired implicant of F needed for the contradiction with Lemma 7.3(ii).

Let $F_1, \ldots, F_a$ be the functions computed by the Σ_2-subcircuits of S that are inputs to the AND gate which is the output gate of S. The function F computed by S can be represented in the form

$$F = \bigwedge_{i=1}^{a} F_i \,.$$

Let A_t denote the set of indices of the functions F_i which are not implied by m_t, i.e. $i \in A_t$ iff m_t is not an implicant of F_i.

Claim 7.5. *There is always a choice of x_t in order to make*

$$|A_t| \leq \frac{|A_{t-1}|}{k/2s} \,.$$

Proof. We describe a choice of x_t that makes A_t sufficiently small. For every i in A_{t-1} we choose some $m_i \in PI\,(F_i)$ with $p_t \supseteq m_i$. Every F_i has such a prime implicant because p_t is a prime implicant of $h = F$. As x_t, we pick a variable of p_t that does not belong to any other of the prime implicants $p_1, \ldots, p_r$. Since each of the prime implicants can share at most $s-1$ variables with each of the other $r-1$ prime implicants, the prime implicant p_t has at least $k - (s-1)\,(r-1)$ variables which do not belong to any of the other prime implicants. Of these "private" variables of p_t, at most $s-1$ can belong to some particular monom m_i we chose, since the circuit has a bottom fanin of at most $s-1$. If we add all the occurrences of the private variables of p_t in the monoms m_i together, we count at most $(s-1)\,|A_{t-1}|$ occurrences. Using that p_t has at least $k - (s-1)\,(r-1)$ private variables, we find that at least one of these variables is in not more than

$$\frac{(s-1)\,|A_{t-1}|}{k - (s-1)\,(r-1)} \leq \frac{|A_{t-1}|}{k/2s}$$

of the chosen monoms. This sufficiently "rare" variable is our choice of x_t. Since only those $i \in A_{t-1}$ remain in A_t for which x_t belongs to the chosen monom m_i, the desired bound for $|A_t|$ follows. □

We now finish the proof of Lemma 7.4. We start with $|A_0| = a < (k/2s)^r$. According to the claim, we can always choose the $x_1, \ldots, x_r$ such that A_r is empty. This means the finally constructed monom m_r is in fact an implicant of F. □

Since Σ_4-circuits can be broken up naturally into Π_3-circuits, our lower bound for monotone Σ_4-circuits follows easily from the previous lemma about monotone Π_3-circuits. We only have to pay attention to a few technicalities.

Theorem 7.6. *Let f be a monotone k-homogeneous s-disjoint function such that $|PI(f)| \leq (k/2s)^{k/2s}$. Then every monotone Σ_4-circuit for f must have at least $|PI(f)| - 1$ OR gates.*

Proof. Let S be a monotone Σ_4-circuit with gates of fanin 2 which computes a monotone k-homogeneous s-disjoint function f. We assume that S has the smallest possible number of OR gates.

Without loss of generality we can assume that no Π_1-subcircuit of S depends on more than $s-1$ variables, i.e. S has bottom fanin at most $s-1$ when regarded as a circuit of unbounded fanin. We can do so because a monom m computed by a Π_1-subcircuit with more than $s-1$ variables can be implied by at most one prime implicant $p \in PI(f)$ (f is s-disjoint). We can remove this Π_1-subcircuit and, if m is implied by $p \in PI(f)$, add p to the top level disjunction of S. The function computed after the modification has the same prime implicants as the original one. This modification is allowed because it leaves the total number of OR gates (with fanin two) unchanged, and we are only interested in this number.

The function f can be represented as a disjunction of functions f_i which are computed by Π_3-circuits: $f = \bigvee f_i$. Let f_i be computed by the Π_3-circuit S_i. Every prime implicant of f must be a prime implicant of at least one of the f_i. Let R be the largest number of prime implicants of f that are prime implicants of one particular $f_i = h$. Let h be computed by the Π_3-circuit $S_i = H$.

We claim that $2 \leq R < k/2s$ cannot hold. To see this, assume the contrary. View H as a Π_3-circuit of unbounded fanin. We can apply Lemma 7.4 to H. Lemma 7.4 yields that H must have a top fanin of at least $(k/2s)^R \geq (k/2s)^2 \geq R^2$. Note that we may assume w.l.o.g. that at most one of the inputs to the top level conjunction of H computes a monom. (We can replace several such inputs by one input computing the conjunction of the monoms.) Using this assumption, we conclude that H requires at least $R^2 - 1$ OR gates. However, a plain disjunction (DNF) of the prime implicants that h shares with f could do the same job that H does in S, and requires only $R - 1 < R^2 - 1$ OR gates of fanin 2. This contradicts our assumption that S has the smallest possible number of OR gates.

To finish the proof of Theorem 7.6, we distinguish the two remaining cases.

Case 1: $R = 1$. Then S is essentially a DNF and needs $|PI(f)| - 1$ OR gates.

Case 2: $R \geq k/2s$. Again we view H as a circuit with gates of unbounded fanin. Applying Lemma 7.4 with $r = k/2s$ yields that H must have a top fanin of at least $(k/2s)^{k/2s} \geq |PI(f)|$ (this inequality is stated as an assumption of the theorem). When built of fanin-2 gates, H requires at least $|PI(f)| - 1$ OR gates since again we may assume w.l.o.g. that at most one of the inputs to the top level conjunction of H consists of a single monom. □

The function $POLY(q, s)$ is q-homogeneous. This function is also s-disjoint because the graphs of two distinct polynomials of degree at most $s - 1$ cannot

share s points. This together with $|PI\,(POLY\,(q,s))| = q^s$ and Theorem 7.6 leads to the following lower bound.

Corollary 7.7. *If $s \leq \sqrt{q}/2$, then any monotone Σ_4-circuit for $POLY\,(q,s)$ must have at least $q^s - 1$ OR gates (just as many as the DNF of this function).*

Chapter 8

Lower Bounds for Pseudoslice Functions

We defined slice functions in Section 3.2. Slice functions are interesting because a superpolynomial lower bound on the monotone complexity of a slice function implies a superpolynomial lower bound on its non-monotone complexity. However, the method of approximations, which is the only known method for proving superpolynomial monotone lower bounds, seems to fail for slice functions. As was explained in Section 3.3, the method of approximations relies on adequate sets of inputs which are mapped to 0 by the function considered. We called them negative test graphs. One property of t-slice functions which seems to make the known arguments unsuitable for them is that they accept *all* inputs with more than t ones. We suggest to approach this problem by studying the complexity of functions that are similar to slice functions. In this chapter we consider functions of the form $f \vee T^n_{t+1}$ that accept all inputs with more than t ones, as does a t-slice function.

Definition 8.1. The *t-pseudoslice function* of f is the function $f'_t = f \vee T^n_{t+1}$.

Let $|x|$ denote the number of ones in the Boolean vector x. Then an equivalent definition of the t-pseudoslice f'_t of f is

$$f'_t(x) = \begin{cases} f(x) & \text{for } |x| < t \\ f(x) & \text{for } |x| = t \\ 1 & \text{for } |x| > t \end{cases}.$$

On the other hand, the t-slice f_t of f assumes the following values:

$$f_t(x) = \begin{cases} 0 & \text{for } |x| < t \\ f(x) & \text{for } |x| = t \\ 1 & \text{for } |x| > t \end{cases}.$$

Thus, the t-slice and the t-pseudoslice function of f only differ for arguments with less than t ones.

We are able to show that our lower bounds for multilinear circuits and monotone Σ_4-circuits also hold for certain pseudoslice functions. We first show how to extend our lower bound for multilinear circuits to pseudoslice functions. The key step is to prove a variant of Lemma 5.8(ii) for pseudoslice functions.

In the proof of Lemma 5.8(ii), the union-freeness of f is only used to assert that the union of the prime implicants p and p' does not contain any new prime implicant. This allows us to state the following generalization of Lemma 5.8(ii).

Corollary 8.2 (Generalization of Lemma 5.8(ii)). *Let g be a gate in a monotone multilinear circuit for a function f and p, p' be prime implicants in $PI_g(f)$. Let $m \subseteq p$ and $m' \subseteq p'$ be distinct prime implicants in $PI(g)$. If f has no prime implicant other than p and p' which is a subset of $p \cup p'$, then the identity $p = (p' \setminus m') \cup m$ holds.*

With this corollary, it is easy to prove a variant of Lemma 5.8(ii) for pseudoslice functions.

Lemma 8.3 (Exchange Lemma for Pseudoslice Functions). *Let g be a gate in a monotone multilinear circuit for the t-pseudoslice f'_t of a monotone k-homogeneous union-free function f such that $t \geq 2k$. Let p and p' be prime implicants of f that are in $PI_g(f'_t)$. If $m \subseteq p$ and $m' \subseteq p'$ are distinct prime implicants in $PI(g)$, then the identity $p = (p' \setminus m') \cup m$ holds.*

Proof. We apply corollary 8.2 to f'_t. According to corollary 8.2, all we need to show is that f'_t has no prime implicant other than p and p' which is a subset of $p \cup p'$. The prime implicants of f'_t are the prime implicants of f, which have length k, and the prime implicants of T^n_{t+1}, which have length $t+1 \geq 2k+1$. Let $q \subseteq p \cup p'$ be a prime implicant of f'_t. We have $|p| = |p'| = k$ because f is k-homogeneous. We conclude $2k \geq |p \cup p'| \geq |q|$. Thus, the prime implicant q of f'_t must also be a prime implicant of f. Because f is union-free, this implies $q = p$ or $q = p'$. □

This lemma easily yields the lower bound for pseudoslice functions.

Theorem 8.4. *Let f be a monotone k-homogeneous union-free function. Then any multilinear circuit which computes the t-pseudoslice of f such that $t \geq 2k$ must have at least $|PI(f)| - 1$ OR gates (just as many as the DNF of this function).*

Proof. We adapt the proof of Lemma 5.9. The idea is to ignore the long prime implicants of the pseudoslice functions. We use Lemma 8.3 in place of Lemma 5.8(ii). This yields the following imitations of Claim 5.10 and Claim 5.11:

Claim 8.5. *Let g be an AND gate in a monotone multilinear circuit for the t-pseudoslice f'_t of f with inputs g_1 and g_2. If $PI(g_1)$ is not empty, then there exists a monom m in $PI(g_1) \cup PI(g_2)$ such that $m \subseteq p$ for all prime implicants $p \in PI(f) \cap PI_g(f'_t)$.*

Claim 8.6. *Let g be a gate in a monotone multilinear circuit for the t-pseudoslice f'_t of f such that $PI(g) \neq \emptyset$. Then*

(i) there is some m in $PI(g)$ such that $m \subseteq p$ for all prime implicants $p \in PI(f) \cap PI_g(f'_t)$, or

(ii) there is some path w from g to the output that is consistent with all prime implicants $p \in PI(f) \cap PI_g(f'_t)$.

Using the same kind of circuit modifications as in the proof of Lemma 5.9, we are able to transform the original circuit for f'_t into a broom-like formula for a function $\widetilde{f}$ such that $PI\left(\widetilde{f}\right) \supseteq PI(f)$. The lower bound then follows with Lemma 5.12. □

Since the function $CLIQUE(n,s)$ is $s(s-1)/2$-homogeneous, we obtain the following lower bound.

Corollary 8.7. *Any multilinear circuit which computes the t-pseudoslice of $CLIQUE(n,s)$ such that $t \geq s(s-1)$ must have at least $\binom{n}{s} - 1$ OR gates.*

The function $POLY(q,s)$ is by definition q-homogeneous.

Corollary 8.8. *Any multilinear circuit which computes the t-pseudoslice of $POLY(q,s)$ such that $s \leq q/2$ and $t \geq 2q$ must have at least $q^s - 1$ OR gates.*

Next we show that our lower bounds for monotone Σ_4-circuits also hold for certain pseudoslices.

Theorem 8.9. *Let f'_t be the t-pseudoslice of a monotone k-homogeneous s-disjoint function f such that $|PI(f)| \leq (k/2s)^{k/2s}$ and $t \geq k^2/2s$. Then every monotone Σ_4-circuit for f'_t must have at least $|PI(f)| - 1$ OR gates.*

Proof. First we prove a version of Lemma 7.4 that also holds for functions $\widetilde{f}$ whose prime implicants are the prime implicants of f and perhaps some additional prime implicants of length more than t. We can proceed as in the proof of Lemma 7.4. We only need to deal with prime implicants of f. We determine an implicant m_r of the given Π_3-circuit for $\widetilde{f}$ in the same way. This implicant has at most $rk \leq k^2/2s$ variables. Hence, the monom m_r must also be an implicant of f, and we again find a contradiction with Lemma 7.3(ii).

We now adapt the proof of Theorem 7.6 in order to make it work for the pseudoslices we are dealing with here. We can basically leave it unchanged. Again, we only deal with prime implicants of f. In the proof of Theorem 7.6 we assume w.l.o.g. that no Π_1-subcircuit depends on more than $s-1$ variables. We give instructions there for modifying the circuit to make it meet this requirement. In the case of computing pseudoslices, these modifications may alter the function computed by the circuit, but we always preserve the prime implicants of f. As a result, we obtain a Σ_4-circuit that computes a function $\widetilde{f}$ as described above. So we can apply the modified version of Lemma 7.4 in the same way as we applied Lemma 7.4 in the proof of Theorem 7.6. This yields the lower bound. □

Corollary 8.10. *Any monotone Σ_4-circuit which computes the t-pseudoslice of $POLY\,(q,s)$ such that $s \leq \sqrt{q}/2$ and $t \geq q^2/2s$ must have at least $q^s - 1$ OR gates.*

Chapter 9

Conclusion

We prove optimal lower bounds on the number of OR gates for multilinear circuits and monotone Σ_4-circuits. These kinds of circuits need as many OR gates as the DNFs of the functions considered. This incompressibility is an interesting property of the functions we study here, namely the clique function and the polynomial function. When dealing with more general circuit models, this may make it easier to prove lower bounds for the clique function and the polynomial function. We give an upper bound for the clique function in order to show that monotone circuits for the clique function require less OR gates than the respective DNFs in general. Hence, our incompressibility results for multilinear circuits computing the clique function cannot be extended to unrestricted monotone circuits. While our upper bound for the clique function also holds for monotone Σ_4-circuits, we give a class of polynomial functions whose monotone Σ_4-circuits are also incompressible. Thus, these polynomial functions are in this sense even harder to compute than clique functions. This observation makes the polynomial function interesting to study when looking for new lower bounds. It is an open problem to find a non-trivial upper bound for the polynomial function. Finally, we note that our lower bounds for multilinear circuits and monotone Σ_4-circuits also hold for certain pseudoslice functions. Since known lower bound arguments for unrestricted monotone circuits seem to fail for pseudoslice functions, our lower bounds could be a starting point for the improvement of lower bounds for unrestricted monotone circuits.

Bibliography

[1] N. Alon and R. B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[2] A. Andreev. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Sov. Math., Dokl.*, 31:530–534, 1985.

[3] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[4] C. Berg and S. Ulfberg. Symmetric approximation arguments for monotone lower bounds without sunflowers. *Comput. Complexity*, 8(1):1–20, 1999.

[5] S. Berkowitz. On some relationships between monotone and non-monotone circuit complexity. Technical report, University of Toronto, 1982.

[6] A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Comput. Complexity*, 3:1–18, 1993.

[7] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[8] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[9] P. E. Dunne. *The complexity of Boolean networks*, volume 29 of *APIC Studies in Data Processing*. Academic Press Ltd., London, 1988.

[10] P. E. S. Dunne. The complexity of central slice functions. *Theor. Comput. Sci.*, 44:247–257, 1986.

[11] M. R. Garey and D. S. Johnson. *Computers and intractability.* W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[12] M. Grigni and M. Sipser. Monotone complexity. In M. S. Paterson, editor, *Boolean function complexity*, volume 169 of *London Mathematical Society Lecture Note Series*, pages 57–75. Cambridge University Press, Cambridge, 1992.

[13] A. Haken. Counting bottlenecks to show monotone $P \neq NP$. In *36th Annual symposium on Foundations of computer science. Held in Milwaukee, WI, USA, October 23-25, 1995. Los Alamitos, CA: IEEE Computer Society Press. 36-40* . 1995.

[14] Hastad. Almost optimal lower bounds for small depth circuits. *ADVCR: Advances in Computing Research*, 5, 1989.

[15] J. Håstad, S. Jukna, and P. Pudlák. Top-down lower bounds for depth-three circuits. *Computational Complexity*, 5(2):99–112, 1995.

[16] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[17] S. Jukna. A note on read-k times branching programs. *RAIRO Inform. Théor. Appl.*, 29(1):75–83, 1995.

[18] S. Jukna. Combinatorics of monotone computations. *Combinatorica*, 19(1):65–85, 1999.

[19] M. Krause. Exponential lower bounds on the complexity of local and real-time branching programs. *J. Inform. Process. Cybernet.*, 24(3):99–110, 1988.

[20] M. Krause, C. Meinel, and S. Waack. Separating the eraser Turing machine classes $\mathrm{L_e}$, $\mathrm{NL_e}$, co-$\mathrm{NL_e}$ and $\mathrm{P_e}$. *Theoret. Comput. Sci.*, 86(2):267–275, 1991.

[21] M. P. Krieger. On the incompressibility of monotone DNFs. Accepted for publication in Theory of Computing Systems.

[22] M. P. Krieger. On the incompressibility of monotone DNFs. In M. Liskiewicz and R. Reischuk, editors, *FCT*, volume 3623 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 2005.

[23] D. E. Muller and F. P. Preparata. Bounds to complexities of networks for sorting and for switching. *J. ACM*, 22(2):195–201, 1975.

[24] N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1996/97.

[25] E. A. Okol′nishnikova. Lower bounds on complexity for the realization of characteristic functions of binary codes by binary programs. *Metody Diskret. Analiz.*, (51):61–83, 113–114, 1991.

[26] N. Pippenger and M. J. Fischer. Relations among complexity measures. *J. Assoc. Comput. Mach.*, 26(2):361–381, 1979.

[27] A. K. Ponnuswami and H. Venkateswaran. Monotone multilinear boolean circuits for bipartite perfect matching require exponential size. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 460–468. Springer, 2004.

[28] S. Ponzio. A lower bound for integer multiplication with read-once branching programs. *SIAM J. Comput*, 28(3):798–815, 1998.

[29] P. Pudlák. The hierarchy of Boolean circuits. *Comput. Artificial Intelligence*, 6(5):449–468, 1987.

[30] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In L. Babai, editor, *STOC*, pages 633–641. ACM, 2004.

[31] R. Raz. Multilinear-$NC_1 \neq$ Multilinear-NC_2. In *FOCS*, pages 344–351. IEEE Computer Society, 2004.

[32] A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Sov. Math., Dokl.*, 31:354–357, 1985.

[33] A. Razborov. Lower bounds on monotone complexity of the logical permanent. *Math. Notes*, 37:485–493, 1985.

[34] A. A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Fundamentals of computation theory (Gosen, 1991)*, volume 529 of *Lecture Notes in Comput. Sci.*, pages 47–60. Springer, Berlin, 1991.

[35] A. A. Razborov and S. Rudich. Natural proofs. *J. Comput. System Sci.*, 55(1, part 1):24–35, 1997. 26th Annual ACM Symposium on the Theory of Computing (STOC '94) (Montreal, PQ, 1994).

[36] J. E. Savage. Computational work and time on finite machines. *J. Assoc. Comput. Mach.*, 19:660–674, 1972.

[37] J. E. Savage. *Models of computation: Exploring the power of computing.* Addison-Wesley Publishing Company, Reading, MA, 1998.

[38] C.-P. Schnorr. The network complexity and the Turing machine complexity of finite functions. *Acta Informat.*, 7(1):95–107, 1976.

[39] C.-P. Schnorr. A Gödel theorem on network complexity lower bounds. *Z. Math. Logik Grundlag. Math.*, 32(4):377–384, 1986.

[40] R. Sengupta and H. Venkateswaran. Multilinearity can be exponentially restrictive (preliminary version). Technical Report GIT-CC-94-40, Georgia Institute of Technology. College of Computing, 1994.

[41] D. Sieling and I. Wegener. Graph driven BDDs—a new data structure for Boolean functions. *Theoret. Comput. Sci.*, 141(1-2):283–310, 1995.

[42] É. Tardos. The gap between monotone and nonmonotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

[43] L. G. Valiant. Negation is powerless for Boolean slice functions. *SIAM J. Comput.*, 15(2):531–535, 1986.

[44] J. H. van Lint. *Introduction to coding theory*, volume 86 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1982.

[45] I. Wegener. On the complexity of slice functions. *Theoret. Comput. Sci.*, 38(1):55–68, 1985.

[46] I. Wegener. *The complexity of Boolean functions.* Wiley-Teubner Series in Computer Science. John Wiley & Sons Ltd., Chichester, 1987.

[47] I. Wegener. *Branching programs and binary decision diagrams. Theory and applications.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

Appendix A

The Method of Approximations

In this appendix we give the proofs that have been omitted in Section 3.3.

Theorem 3.13 ([32, 1]). *For every monotone function f and every legitimate lattice $\mathcal{K}$, we have $C^+(f) \geq \varrho(f, \mathcal{K})$.*

Proof. Let S be a monotone circuit for f. We show that the distance of f to $\mathcal{K}$ does not exceed the number of gates of S. To bound the distance of f to $\mathcal{K}$, we consider the approximator circuit S' we obtain by replacing the gates of S by corresponding lattice operations. An AND gate is replaced by $\sqcap$, an OR gate is replaced by $\sqcup$. The inputs to S' are the lattice elements $A(x_1), \ldots, A(x_n)$ that correspond to the variables $x_1, \ldots, x_n$.

To prove the theorem, we need to find a lattice element A and pairs (A_1, B_1), $\ldots, (A_t, B_t)$ of lattice elements such that equations (3.1) and (3.2) hold. As A, we choose the lattice element computed by S'. Through a topological sort of S, we arrange the gates of S in an order $g_1, \ldots, g_t$ such that every input to some gate g_i is either a variable or is computed by some gate g_j with $j < i$. As A_i and B_i, we choose the inputs to the lattice operation in S' that corresponds to gate g_i of S.

We now prove a more general claim by induction.

Claim A.1. *For all i, $1 \leq i \leq t$, if f is the function computed by the gate g_i in S and if A is the result of the lattice operation corresponding to g_i, the equations (3.1) and (3.2) hold.*

Proof. We prove the claim by induction on t. For $t = 0$, the function f is a variable, and $A = A(f)$. Assume that the claim holds for $t - 1$. We now show that the equations hold for $i = t$. We consider the case that g_t is an OR gate. Let f_t and h_t be the functions computed at the inputs to g_t. By the induction hypothesis, we have

$$A_t \subseteq A(f_t) \cup \bigcup_{i=1}^{t-1} \delta_{\sqcup}(A_i, B_i)$$

and

$$B_t \subseteq A(h_t) \cup \bigcup_{i=1}^{t-1} \delta_{\sqcup}(A_i, B_i) \ .$$

So if A is the result of the lattice operation corresponding to g_i, then

$$\begin{aligned} A &= A_t \sqcup B_t = A_t \cup B_t \cup \delta_{\sqcup}(A_t, B_t) \\ &\subseteq A(f_t) \cup A(h_t) \cup \bigcup_{i=1}^{t} \delta_{\sqcup}(A_i, B_i) \\ &\subseteq A(f) \cup \bigcup_{i=1}^{t} \delta_{\sqcup}(A_i, B_i) \ , \end{aligned}$$

so equation (3.1) holds. For the other equation, we note that the induction hypothesis yields

$$A(f_t) \subseteq A_t \cup \bigcup_{i=1}^{t-1} \delta_{\sqcap}(A_i, B_i)$$

and

$$A(h_t) \subseteq B_t \cup \bigcup_{i=1}^{t-1} \delta_{\sqcap}(A_i, B_i) \ .$$

So if A is the result of the lattice operation corresponding to g_i, then

$$\begin{aligned} A(f) &= A(f_t) \cup A(h_t) \\ &\subseteq A_t \cup B_t \cup \bigcup_{i=1}^{t-1} \delta_{\sqcap}(A_i, B_i) \\ &\subseteq A_t \sqcup B_t \cup \bigcup_{i=1}^{t-1} \delta_{\sqcap}(A_i, B_i) \\ &= A \cup \bigcup_{i=1}^{t-1} \delta_{\sqcap}(A_i, B_i) \\ &\subseteq A \cup \bigcup_{i=1}^{t} \delta_{\sqcap}(A_i, B_i) \ , \end{aligned}$$

so equation (3.2) holds too.

The case that g_t is an AND gate is treated similarly. □

The proof of the claim finishes the proof of the theorem. □

Lemma 3.14. *For all $r \geq 2$ and $k \geq 0$, we have $h(r,k) \leq (r-1)^k$.*

Proof. We prove the lemma by induction on r. For $r = 2$, it is easy to see that $h(r,k) = 1$. Suppose that $\mathcal{F}$ contains two sets W_1 and W_2. We set $U := W_1 \cap W_2$. If $U \subsetneq W_1$, then we set $W = W_1$. Otherwise, we have $U \subsetneq W_2$, and we set $W = W_1$. In any case we see that $\mathcal{F}$ does not have property $P(r,k)$.

Assume that the bound holds for $r-1$. We now prove it for r. Let $\mathcal{F}$ have property $P(r,k)$. We pick some arbitrary set $D \in \mathcal{F}$. For every subset $C \subseteq D$, we define

$$\mathcal{F}_C := \{W - C \mid W \in \mathcal{F} \wedge W \cap D = C\} .$$

We claim that $\mathcal{F}_C$ has property $P(r-1, k-|C|)$. For the sake of contradiction, suppose that there are $W', W'_1, \ldots, W'_{r-1} \in \mathcal{F}_C$ and $U' \subsetneq W'$ such that $W'_i \cap W'_j \subseteq U'$ for all $1 \leq i < j \leq r-1$. Then we can define $W := W' \cup C$, $U := U' \cup C$, $W_i := W'_i \cup C$ (for $1 \leq i \leq r-1$) and $W_r := D$. These sets satisfy $U \subsetneq W$ and $W_i \cap W_j \subseteq U$ for all $1 \leq i < j \leq r$. This contradicts the fact that $\mathcal{F}$ has property $P(r,k)$.

Using that $\mathcal{F}_C$ has property $P(r-1, k-|C|)$ we can now prove the bound on $h(r,k)$ for r. By the inductive hypothesis, we have $|\mathcal{F}_C| \leq (r-2)^{k-|C|}$. As a result,

$$\begin{aligned} |\mathcal{F}| &= \sum_{C \subseteq D} |\mathcal{F}_C| \leq \sum_{C \subseteq D} (r-2)^{k-|C|} \\ &= \sum_{i=0}^{|D|} \binom{|D|}{i} (r-2)^{k-i} . \end{aligned}$$

Since D has cardinality at most k, we have

$$|\mathcal{F}| \leq \sum_{i=0}^{k} \binom{k}{i} (r-2)^{k-i} .$$

Then the Binomial Theorem yields the desired bound $|\mathcal{F}| \leq (r-1)^k$. □

A.1 Lemmas for Handling the Clique Function

Lemma 3.17. *Let $A \subseteq \mathcal{V}(l)$, $A \vdash W$ and O be a random $(s-1)$-coloring of all vertices. Then*

$$\mathbb{P}[W \text{ is properly colored by } O \text{ and no set in } A \text{ is properly colored by } O] \leq \left(1 - \frac{(s-1)(s-2)\cdots(s-l)}{(s-1)^l}\right)^r .$$

Proof. By definition, if $A \vdash W$, then there are $W_1, \ldots, W_r \in A$ such that $W_1, \ldots, W_r \vdash W$. For particular sets $W_1, \ldots, W_r$ with this property we have the inequalities

$$\begin{aligned}&\mathbb{P}\left[W \text{ is properly colored and no set in } A \text{ is properly colored}\right]\\&\qquad\leq \mathbb{P}\left[W \text{ is properly colored and } W_1,\ldots,W_r \text{ are not properly colored}\right]\\&\qquad\leq \mathbb{P}\left[W_1,\ldots,W_r \text{ are not properly colored} \,|\, W \text{ is properly colored}\right].\end{aligned}$$

If this probability is not 0, the sets $W_1,\ldots,W_r$ are distinct because if $W_i = W_j$, then $W \supseteq W_i$, and W_i is properly colored if W is. So if the probability is not 0, the events $\{W_i \text{ is not properly colored} \,|\, W \text{ is properly colored}\}$ are independent because $W_i \cap W_j \subseteq W$ for $i \neq j$. This leads to

$$\begin{aligned}&\mathbb{P}\left[W_1,\ldots,W_r \text{ are not properly colored} \,|\, W \text{ is properly colored}\right]\\&\qquad=\prod_{i=1}^{r}\mathbb{P}\left[W_i \text{ is not properly colored} \,|\, W \text{ is properly colored}\right].\end{aligned}$$

We set $p_i := |W_i \cap W|$ and $q_i := |W_i \setminus W|$. This gives us

$$\begin{aligned}&\mathbb{P}\left[W_i \text{ is not properly colored} \,|\, W \text{ is properly colored}\right]\\&\qquad=1-\mathbb{P}\left[W_i \text{ is properly colored} \,|\, W \text{ is properly colored}\right]\\&\qquad=1-\frac{(s-1-p_i)\,(s-1-p_i-2)\cdots(s-p_i-q_i)}{(s-1)^{q_i}}.\end{aligned}$$

By using $p_i + q_i = |W| \leq l$, we obtain

$$\begin{aligned}&1-\frac{(s-1-p_i)\,(s-1-p_i-2)\cdots(s-p_i-q_i)}{(s-1)^{q_i}}\\&\qquad\leq 1-\frac{(s-1-p_i)\,(s-1-p_i-2)\cdots(s-l)}{(s-1)^{l-p_i}}\\&\qquad\leq 1-\frac{(s-1)\,(s-2)\cdots(s-l)}{(s-1)^{l}}.\end{aligned}$$

Altogether we obtain the desired bound. □

A.2 Lemmas for Handling the Polynomial Function

Lemma 3.20. *If* $r \leq q/3+1$, *then*

$$\delta_+(q,r,l) \leq 3q^{s-\lceil (l+1)/2\rceil}\,(r-1)^{\lceil (l+1)/2\rceil}.$$

Proof. By definition, for lattice elements $M, N \in \mathcal{K}(q, r, l)$ we have

$$\begin{aligned} \delta_{\sqcap}(M, N) &= (\lceil A \rceil \cap \lceil B \rceil) \setminus (\lceil A \rceil \sqcap \lceil B \rceil) \\ &= (\lceil A \rceil \cap \lceil B \rceil) \setminus (\lceil A \cap B \rceil) \end{aligned}$$

for closed subsets $A, B \subseteq \mathcal{E}(l)$. Consider an element b in $\lceil A \rceil \cap \lceil B \rceil$. There must be minimal elements $F_1 \in A$ and $F_2 \in B$ such that b has ones in all entries corresponding to the elements in $F_1 \cup F_2$. If $|F_1 \cup F_2| \leq l$, then b is in $\lceil A \cap B \rceil$, so $b \notin \delta_{\sqcap}(\lceil A \rceil, \lceil B \rceil)$. We conclude that for every element $b \in \delta_{\sqcap}(\lceil A \rceil, \lceil B \rceil)$, there is a set F of cardinality $k \geq \lceil (l+1)/2 \rceil$ in $A \cup B$. Since the sets A and B are closed and Corollary 3.15 also holds for closed subsets of $\mathcal{E}(l)$, each of them has at most $(r-1)^k$ minimal elements of cardinality k.

Consider a set $F \subseteq GF(q) \times GF(q)$ in $A \cup B$ with cardinality $|F| = k$. If there are group elements $g_1, g_2, g_3 \in GF(q)$ such that $g_2 \neq g_3$, $(g_1, g_2) \in F$ and $(g_1, g_3) \in F$, then there is no positive test graph which results from F because a polynomial cannot map g_1 to both g_2 and g_3. Hence, we only need to consider sets F that "assign" values to $|F| = k$ group elements. Since a polynomial of degree at most $s-1$ is uniquely determined by images of s elements in its domain, there are q^{s-k} polynomials that comply with the assignments determined by F.

Altogether, for every possible cardinality k, $\lceil (l+1)/2 \rceil \leq k \leq l$, there are at most $2(r-1)^k$ minimal sets in $A \cup B$, of which each can contribute q^{s-k} positive test graphs. We accordingly estimate the number of positive test graphs in $\delta_{\sqcap}(M, N)$ as follows:

$$\begin{aligned} |\delta_{\sqcap}(M, N) \cap E_+| &\leq \sum_{k=\lceil (l+1)/2 \rceil}^{l} 2(r-1)^k q^{s-k} \\ &= 2q^s \sum_{k=\lceil (l+1)/2 \rceil}^{l} \left(\frac{r-1}{q}\right)^k \\ &< 2q^{s-\lceil (l+1)/2 \rceil} (r-1)^{\lceil (l+1)/2 \rceil} \sum_{k=0}^{\infty} \left(\frac{r-1}{q}\right)^k \\ &\leq 2q^{s-\lceil (l+1)/2 \rceil} (r-1)^{\lceil (l+1)/2 \rceil} \sum_{k=0}^{\infty} \left(\frac{1}{3}\right)^k \\ &= 3q^{s-\lceil (l+1)/2 \rceil} (r-1)^{\lceil (l+1)/2 \rceil} . \end{aligned}$$

□

Lemma 3.21. *We have*

$$\delta_{-}(q, r, l) \leq q^{2l} (\varepsilon l)^r .$$

Proof. As for the clique function, we study the properties of implications $W_1, \ldots, W_r \vdash W$ with respect to the random test graph. For the test graph $G \in \mathbb{B}^{q^2}$,

let $E(G)$ denote the subset of $GF(q) \times GF(q)$ whose entries are 1 in G. For $A \subseteq \mathcal{E}(l)$, $A \vdash W$ and particular sets $W_1, \ldots, W_r \in A$ such that $W_1, \ldots, W_r \vdash W$ we have

$$\begin{aligned}\mathbb{P}[W \subseteq E(G) \wedge \forall F \in A\, F \not\subseteq E(G)] \\ &\leq \mathbb{P}[W \subseteq E(G) \wedge W_1 \not\subseteq E(G) \wedge \ldots \wedge W_r \not\subseteq E(G)] \\ &\leq \mathbb{P}[W_1 \not\subseteq E(G) \wedge \ldots \wedge W_r \not\subseteq E(G) \mid W \subseteq E(G)]\,.\end{aligned}$$

If this probability is not 0, the sets $W_1, \ldots, W_r$ are distinct because if $W_i = W_j$, then $W \supseteq W_i$, and $W_i \subseteq E(G)$ if $W \subseteq E(G)$. If $W_1, \ldots, W_r$ are distinct, then, by the definition of the implication, the events $\{W_i \not\subseteq E(G) \mid W \subseteq E(G)\}$ are independent. Exploiting this we conclude

$$\begin{aligned}\mathbb{P}[W_1 \not\subseteq E(G) \wedge \ldots \wedge W_r \not\subseteq E(G) \mid W \subseteq E(G)] \\ &= \prod_{i=1}^{r} \mathbb{P}[W_i \not\subseteq E(G) \mid W \subseteq E(G)]\,.\end{aligned}$$

It is clear that

$$\mathbb{P}[W_i \not\subseteq E(G) \mid W \subseteq E(G)] = 1 - (1-\varepsilon)^{|W_i \setminus W|} \leq 1 - (1-\varepsilon)^l \leq \varepsilon l\,.$$

Altogether we have

$$\mathbb{P}[W \subseteq E(G) \wedge \forall F \in A\, F \not\subseteq E(G)] \leq (\varepsilon l)^r\,. \tag{A.1}$$

By definition,

$$\delta_{\sqcup}(\lceil A \rceil, \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus (\lceil A \rceil \cup \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus \lceil A \cup B \rceil\,.$$

Thus, the event $G \in \delta_{\sqcup}(\lceil A \rceil, \lceil B \rceil) = \lceil (A \cup B)^* \rceil \setminus (\lceil A \rceil \cup \lceil B \rceil)$ only occurs if a subset of $E(G)$ is in $A \cup B$ but no subset of $E(G)$ in $(A \cup B)^*$. We imagine that the closure $(A \cup B)^*$ is constructed from $A \cup B$ by successively adding new sets $W_1, \ldots, W_p$, where $A \cup B \cup \{W_1, \ldots, W_{i-1}\} \vdash W_i$. According to (A.1), the probability of the event

$$\begin{aligned}\{&\text{a subset of } E(G) \text{ is in } A \cup B \cup \{W_1, \ldots, W_i\} \\ &\qquad \text{but no subset of } E(G) \text{ is in } A \cup B \cup \{W_1, \ldots, W_{i-1}\}\}\end{aligned}$$

is not more than $(\varepsilon l)^r$. The number p of sets added is at most $|\mathcal{E}(l)| \leq q^{2l}$. The bound on $\delta_{-}(q, r, l)$ follows. □